Design, Optimization, and Prototyping of a Three Translational Degree of Freedom Parallel Robot

by

Jonathan Hodgins

A Thesis Submitted in Partial Fulfillment

of the Requirements for the Degree of

Master of Applied Science

in

The Faculty of Engineering and Applied Science

Mechanical Engineering Program

University of Ontario Institute of Technology

March, 2012

Abstract

This thesis presents an evolutionarily design change for the Delta parallel robot. The proposed design change increases the useful workspace of the robot and aids in permanently avoiding singularities in the workspace. This is accomplished by means of a new intermediate link parallel to the 4 bar linkage.

The addition of the new link simultaneously increases the total workspace volume and decreases the dexterity without significantly affecting the stiffness.

The design is analyzed and the inverse kinematics, Jacobian, stiffness and dexterity relations are formulated. The relations are then converted into a form that is usable by MATLAB to calculate different workspaces that illustrate the advantages of the new design. Subsequently, an optimization problem is formulated that aims to take advantage of the new attributes to create a balanced robot that further illustrates the benefits of the new design. The results are clearly illustrated by comparing plotted sections of workspace from both the optimized and unoptimized workspace.

Lastly, the design is developed into a 3D model which is then fabricated into a working prototype to test and verify functionality.

Acknowledgments

Foremost I would like to thank my supervisor, Dr. Dan Zhang, for his guidance, without which this thesis couldn't have been completed. He also helped my push the limits of my knowledge and abilities helping me produce better results. I would also like to also thank Dr. Zhang for his financial support during my Master program.

Thanks are also due to the committee members; Dr. Xiaodong Lin, Dr. Hossam Kishawy and Dr. Ghaus Rizvi. Without their expertise this thesis would not be completed to its current level.

I would also like to thank Zhen Gao, Gianmarc Coppola and Zhongzhe Chi for their time and help while I was working my thesis, I couldn't have completed this thesis without their help.

Contents

1 Introduction **1**

1.1 Background 2

1.2 Outline 3

1.3 Contributions 4

2 Design **6**

2.1 Introduction 6

2.2 Geometric Description 7

2.3 Physical Constraints 11

3 Inverse Kinematics and Workspace **12**

3.1 Overview 12

3.2 Inverse Kinematics 12

3.3 Workspace 14

4 Jacobian, Stiffness and Dexterity Analysis **17**

4.1 Introduction 17

4.2 Jacobian 17

4.3 Stiffness Analysis 21

4.4 Dexterity Analysis 23

5 Optimization **26**

5.1 Introduction 26

5.2 Optimization Setup 26

5.3 Optimization Results 29

6 Prototype **34**

6.1 Design 34

6.2 Constructed Prototype 35

7 Additional Features and Future Work **38**

8 Summary **39**

List of Figures

1 Delta Robot Schematic 2
2 University of Maryland Manipulator Schematic 3
3 Delta Parallel Robot Single Arm Schematic 6
4 University of Maryland Manipulator Single Arm Schematic 7
5 Parallel Linkages Assumed Kinematically Equal 8
6 Equivalent Link Layouts 8
7 Proposed structure 9
8 Single New Arm Schematic 10
9 New Arm Schematic Labeled 10
10 Standard Delta Workspace With Unoptimized Link Lengths 14
11 Robot Workspace Where Link C is Varied 15
12 Schematic Labeling for Jacobian 18
13 Slices of Workspace Showing Average Stiffness 22
14 Stiffness Slices Along X-Z Plane 23
15 Slices of Workspace Showing Dexterity 25
16 Dexterity Slices Along X-Z Plane 26
17 MATLAB Optimization Toolbox 29
18 Optimization Function Value Vs Iteration 30
19 Optimization Function Inputs to the Objective Function Over Time . 31
20 Optimized Workspace (Top Trimetric View) 31
21 Optimized Workspace (Bottom Trimetric View) 32
22 Optimized Stiffness and Dexterity 33
23 NX7.5 CAD Model 35
24 NX7.5 CAD Model Adjusted 36
25 Prototype 37
26 Prototype Alternate Positions 37

27 New Single Arm Schematic with additional linkage 38

1 Introduction

The most commonly used robotic manipulator in industry is a serial robot arm. It is a manipulator where the motors and links are connected in one chain to the end effector.

Another option that is becoming more popular is parallel robotic manipulator. They consists of multiple linkages connecting from the base to the end effector. The number of linkages corresponds to the degrees of freedom of the end effector because each arm is typically driven by one actuator. Because of this configuration, there are no heavy parts to move and it has a low inertia allowing the robot to achieve high accelerations.

Parallel robots have many advantages over serial robots, the main advantages being stiffness from the multiple linkages and speed from the low inertia. But parallel manipulators to have some drawbacks, one being that they have a restricted workspace volume compared to a serial robot of similar size. There are many described parallel robots, the first being the attributed to Gough [1] and Stewart [2]. There are many possible variations of parallel robots [3], each with advantages and disadvantages.

A parallel robot that has seen commercial success is the Delta robot [4, 5, 6], shown in Fig. 1. The Delta robot is well established in industry as a good pick and place machine for relatively light loads. It is well suited to this application because it has a workspace suited to move objects horizontally, and it can be designed with very low link inertia for high accelerations. The inertia is kept low because as a parallel robot, it does not need to move the weight of the motors, meaning they can move quickly with less force. A variant of the Delta robot has even been designed to maximize speed [7]. Because the Delta robot only has translation degrees of freedom, there are only 3 motors, which makes it easier to control as well as maintain.

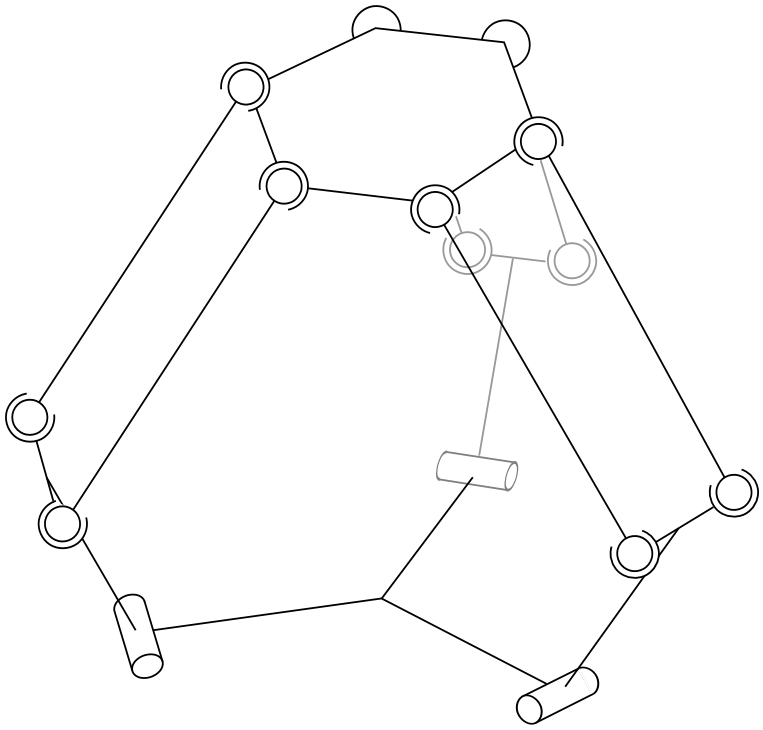


Figure 1: Delta Robot Schematic

1.1 Background

Since the Delta robot has seen success, there have been many contributions to improve the design. One variant of a Delta robot is called the University of Maryland Manipulator [8]. The University of Maryland Manipulator is initially presented with an extra link that allows the planar 4-bar linkage to be shorter than surrounding two revolute joints, which remain the same distance apart. A three dimensional schematic representation of the University of Maryland Manipulator is provided in Fig. 2. In the thesis, after the manipulator was analyzed and optimized, the new link was optimized to a length of 0, so it essentially becomes a Delta robot constructed using only revolute joints.

Other variants of the Delta robot have replaced the driving rotary actuators with prismatic actuators, changing the workspace shape [9, 10]. One variant has the 3 prismatic actuators parallel to each other, so the volume of the workspace is only limited by the range of motion of the actuators [11].

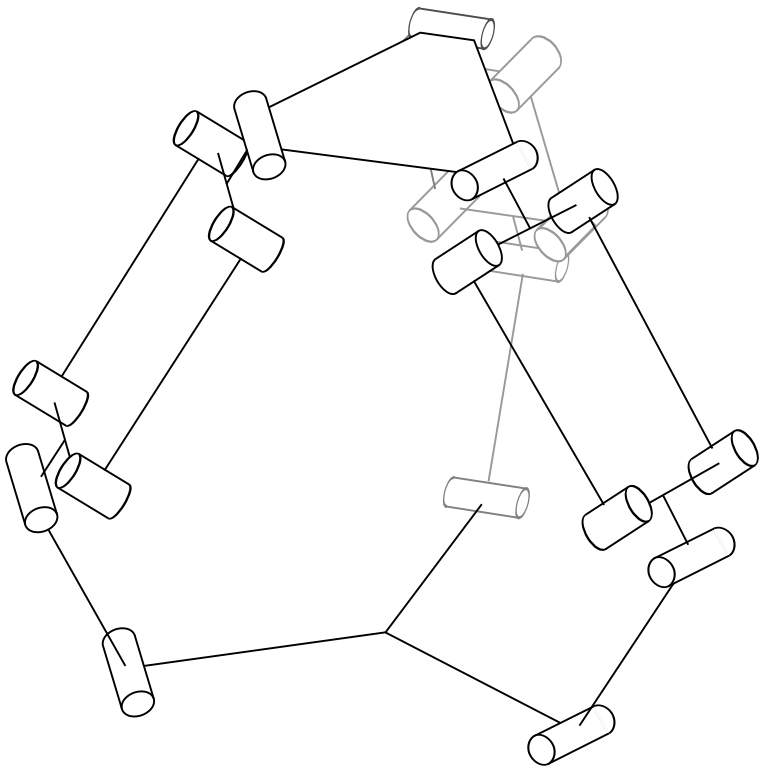


Figure 2: University of Maryland Manipulator Schematic

There have also been some Delta robot variants developed with higher degrees of freedom (DOF). These higher DOF robots use the planar 4-bar linkage in unique configurations [12] as well as breaking up the planar 4-bar linkage to get a full 6 DOF robot [13]. The 6 DOF robot based on the Delta robot separates the driving link of a Delta robot lengthwise, so it now uses 2 motors instead of one. Each motor will drive each side of the 4-bar linkage, so it is now possible for the end effector to twist.

1.2 Outline

This thesis presents the design concept and structure for a novel arrangement of links based on a Delta robot, as well as any assumptions made to achieve the new configuration. The design for a novel configuration to create a 3 degree of freedom translation robot is first outlined in chapter 2. Then the inverse kinematics are formulated in chapter 3 and analyzed by graphing the 3 dimensional workspace. To further analyze

the new structure, the Jacobian is formulated and verified by comparing the results to existing work in chapter 4. The stiffness and dexterity are derived from the Jacobian and programmed in MATLAB, so it is possible to find the average of each for a given workspace.

An optimization problem is then devised that will maximize the workspace, stiffness and minimize the dexterity in chapter 5. The optimization is set up in MATLAB by using the optimization toolbox and creating an objective function to be optimized. The optimizer adjusts 2 link lengths to find the optimal robot configuration.

From the optimized values, a working prototype is subsequently designed in NX 7.5 CAD software, built using a laser cutter and 3D printer, and tested to verify functionality as outlined in chapter 6.

1.3 Contributions

This thesis aims to design and present the next step in the evolution of the Delta robot. The goals of this thesis are to design a new robotic manipulator to have an increased and more useable workspace. Other goals are to maintain or increase the stiffness compared to an equivalent Delta robot, make it simpler to construct, and improve the dexterity by moving any singularities away from the workspace.

The joints in a Delta robot are re-designed to include a new link, parallel to the 4 bar linkage. Furthermore, the new robot is analyzed by formulating the inverse kinematics which is then used to calculate and plot the total reachable workspace. The new workspace is then compared to an equivalent Delta robot to illustrate the advantages of the new link.

The analysis is continued by finding the Jacobian matrix, which relates the end velocity to the joint velocity. Then using the Jacobian, the stiffness and dexterity of the new parallel robot are found. The stiffness and dexterity are important aspects to consider when designing a parallel robot, which is why they are used with the volume

to optimize the new link length as well as the length of the 4 bar linkage.

To verify the analysis, a working prototype is designed, constructed, and programmed to run through a series of patterns. This confirms that the design is a valid and useful parallel robot.

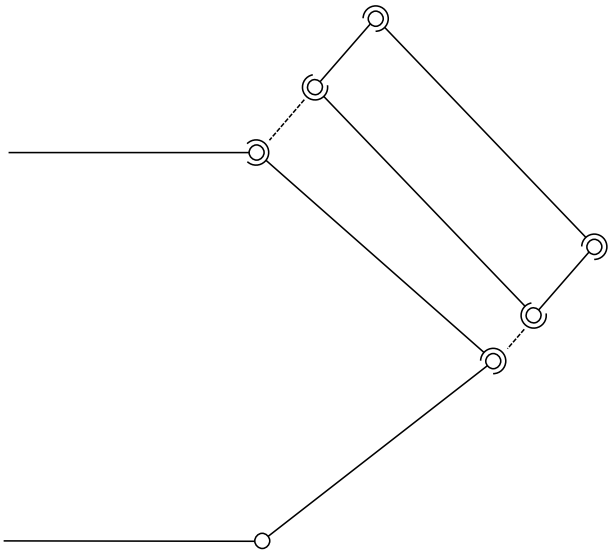


Figure 3: Delta Parallel Robot Single Arm Schematic

2 Design

2.1 Introduction

Figure 3 shows a schematic representation of a single arm of a Delta robot. It shows a side view as well as a projected view of the 4-bar linkage. All following designs will be presented in this format for comparison purposes. The Delta robot consists of 3 of these arms at evenly spaced angles.

The idea for the design presented in this thesis came about while attempting to design a 3-DOF parallel robot that uses only revolute joints. During the design process, a design similar to the University of Maryland Manipulator [5] was developed. The robot presented in this paper is similarly constructed using only revolute joints, which allows it to be more economical to produce. The ball joints traditionally used in Delta robots tend to be less stiff and more expensive.

The main difference between the robot presented in this paper and the University of Maryland Manipulator is the offsetting of the planar 4-bar linkage and a key change in the extra link. Because of how the University of Maryland manipulator linkage is set up, as seen in Fig.4, the 4-bar linkage is always shorter than or equal to the

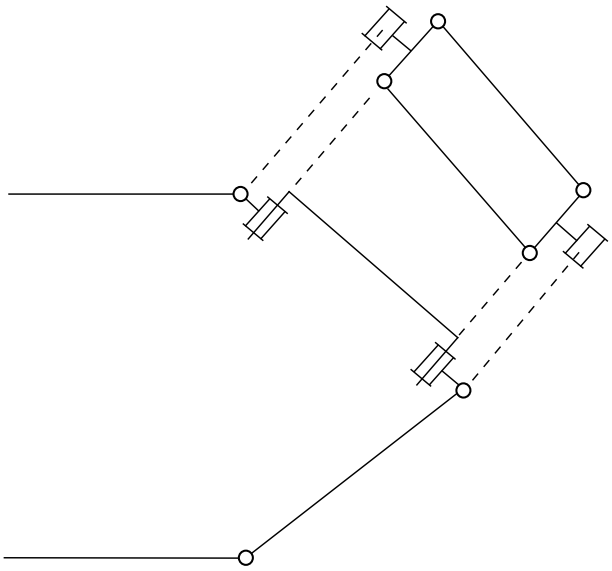


Figure 4: University of Maryland Manipulator Single Arm Schematic

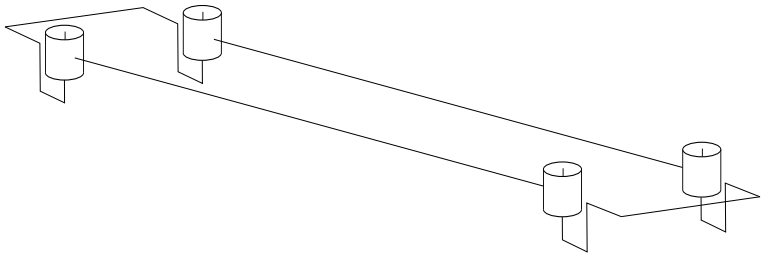
distance between the surrounding revolute joints.

To develop the robot presented in this paper, some assumptions are required about the kinematics of the 4 bar linkage to create the structure proposed in chapter 2.2. The first assumption is that a parallel linkage can be slightly shifted in the direction perpendicular to the links and still be able to bear an axial load without compromising the structure. This is illustrated in Fig. 5, where a comparison between the parallel links in line and not in line is shown.

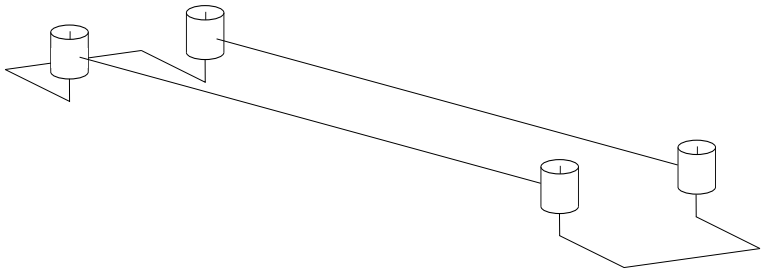
Another assumption was made to simplify the inverse kinematics is detailed in Fig. 6. The two shorter links are added together to make a single link length because they are always parallel, which simplifies the calculations and analysis.

2.2 Geometric Description

The structure of the new parallel robot is shown in Fig. 7 with a the schematic of a single arm given in Fig. 8. In Fig. 8, the base is the link in the lower left of the figure, it is connected to the next link by a driving rotary joint. The next link moving along the chain is the new link. The planar 4-bar linkage connects the end effector



(a) Parallel link where the ends are in line with the parallel links



(b) Parallel link where the ends are not in line with the parallel links

Figure 5: Parallel Linkages Assumed Kinematically Equal

(a) Actual Link Arrangement

(b) Simplified Link Arrangement Where $C = C1 + C2$

Figure 6: Equivalent Link Layouts

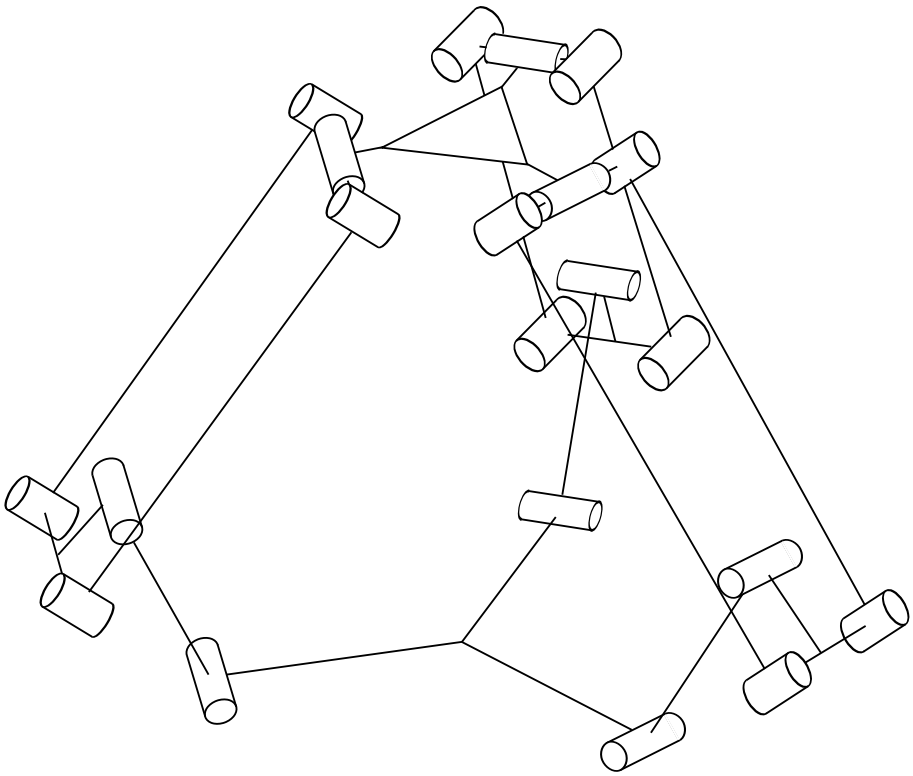


Figure 7: Proposed structure

to the new link with 2 additional rotary joints.

All of the links lengths, end points and required angles from figure 8 are labeled in figure 9. The first joint, point P_a, is the driving joint that moves the arm, so the angle of the link, θ, is calculated in the inverse kinematics. Each end point of a link is labeled as a point with 3 values; an x, y, and z location. Each link length is a constant value labeled from a through e. Length f is the distance from Pb to Pd projected into the plane of the image (y = 0). Each end point of the links are labeled as a 3 dimensional vector, describing its location in space. Each link length is a constant value labeled from a through e. The length f is the calculated distance from Pb to Pd projected into the X-Z plane.

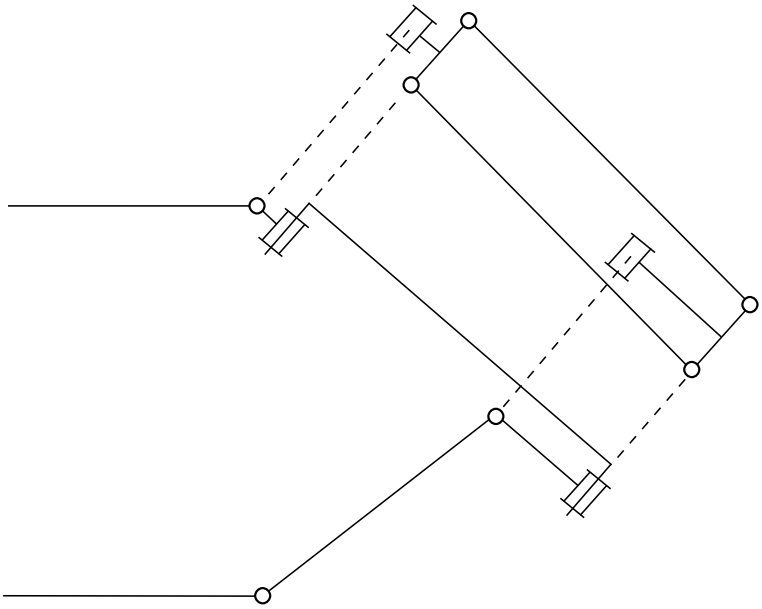


Figure 8: Single New Arm Schematic

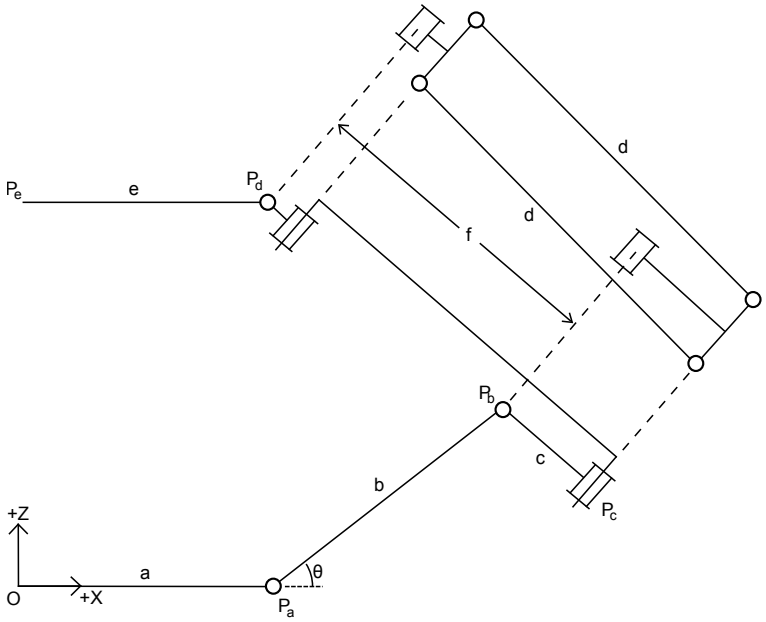


Figure 9: New Arm Schematic Labeled

2.3 Physical Constraints

By identifying and including any physical limitations in the analysis of the robot, it makes the analysis of the theoretical robot more valuable when building the physical robot. This analysis is helpful because the optimized workspace is completely valid and useable, whereas if the limits were left unidentified the software risks damage to the prototype later by over-extending joints.

Table 1: Physical Limits - All reference variables from Fig. 9

Limit	Explanation
Point P_c is prevented from having a higher z value than point P_d	If P_{cz} is above P_{dz} than the arm sticks out into the workspace, potentially colliding with an object in the workspace. Also, and point that the robot could reach when $P_{cz} > P_{dz}$ is unusable because it at an extremity of the workspace.
P_e is considered an invalid position if its Y value is greater than $0.9 * d$	If the Y value of the end effector exceeds this limit, the parallel linkage approaches too close to a singularity to be useful.
The angle θ is limited between 0^{o} and 85^{o}	If $\theta < 0$ then the arm could potentially approach a singularity where b and d are parallel. If $\theta > 85$ then the arm approaches a singularity where the lower arm lines up with the upper arm.

The physical limits are listed with their corresponding explanation in Table 1.

3 Inverse Kinematics and Workspace

3.1 Overview

By deriving the inverse kinematics, which calculate actuator position based on desired end effector position, it is possible to calculate the joint angles. This allows the robot to then know how to position the motors so that the end effector is at a given position. There are multiple ways to approach the inverse kinematics of the presented robot, the inverse kinematic equations presented in chapter 3.2are formulated using a graphical method and verified using the closed loop vector method when the Jacobian is derived in chapter 4.2. All equation labels in the following chapter refer to figure 9. Any lower case letter is a scalar length with units of mm and the points are 3 dimensional coordinates also in mm.

The inverse kinematics can also be used to check if a given coordinate point is reachable by the mechanism, by checking every point within a grid, it is possible to find the shape and size of the workspace, shown in chapter 3.3.

3.2 Inverse Kinematics

The first step to calculate the inverse kinematics is to project the position of the end effector and the 4-bar linkage onto the x-z plane. The length f is the distance from P_b to P_d projected onto the x-z plane.

$$f = \sqrt{d^2 - P_{dy}^2} - c \tag{1}$$

Now, with two known points P_a and P_d as well as 2 radii, b and f respectively, it is possible to calculate where a circle at P_a with radius b and a circle a P_d with radius f intersect, which is point P_b.

The value $dCalc$ is the distance between P_a and P_d. This is given by:

$$dCalc = \sqrt{(P_{dx} - P_{ax})^2 + (P_{dz} - P_{az})^2} \tag{2}$$

If $dCalc < b + f$ and $dCalc > 0$ then the intersection point is valid and can be calculated. These define the constraints on the value of $dCalc$. Subsequently, the distances can be calculated as

$$aCalc = \frac{b^2 - f^2 + dCalc^2}{2 * dCalc} \tag{3}$$

$$hCalc = \sqrt{b^2 - aCalc^2} \tag{4}$$

$$P_{bx} = P_{ax} + aCalc * \frac{P_{dx} - P_{ax}}{dCalc} + hCalc * \frac{P_{dz} - P_{az}}{dCalc} \tag{5}$$

$$P_{by} = 0 \tag{6}$$

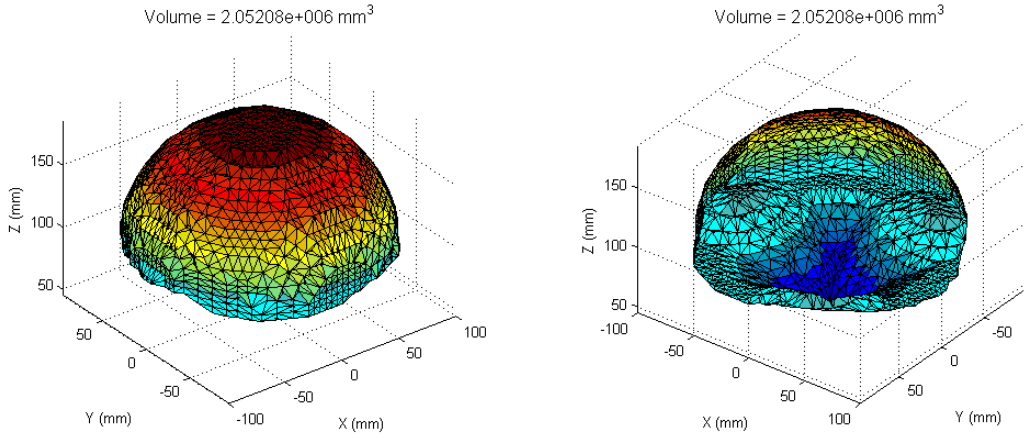
$$P_{bz} = P_{az} + aCalc * \frac{P_{dz} - P_{az}}{dCalc} + hCalc * \frac{P_{dx} - P_{ax}}{dCalc} \tag{7}$$

In the above, there are 2 possible solutions for calculating P_b because the circles have two intersection points. Because of the orientation of the arm, the correct point is always the larger one, thus removing one solution.

With the two endpoints of the driving link known, it is simple to calculate the angle θ.

$$\theta = atan2\left(P_{bz} - P_{az}, P_{bx} - P_{ax}\right) \tag{8}$$

The function atan2 represents the arc-tangent taking into consideration the quadrant.



(a) Delta Workspace Top (b) Delta Workspace Underside

Figure 10: Standard Delta Workspace With Unoptimized Link Lengths

To calculate the angle for the other 2 arms, the simplest solution is to rotate the coordinate system by 120 degrees (the angle between the arms) and perform the same calculations above for each arm.

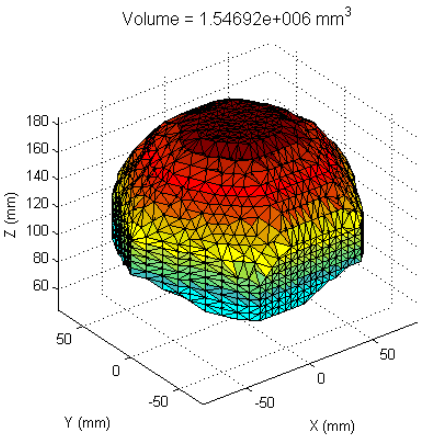
The complete MATLAB code used to calculate the workspace is given in Appendix A.

3.3 Workspace

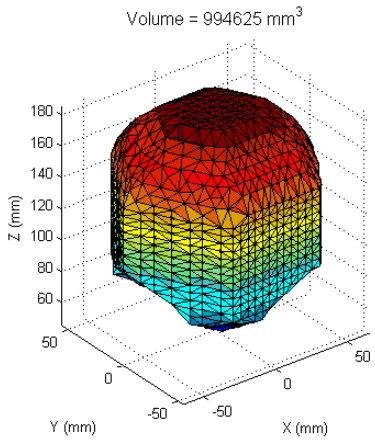
The workspace of an unoptimized Delta robot, as presented in Fig. 3, is shown in Fig. 10. An interesting feature to note is the pattern that the limits of the joints creates in the underside of the workspace, one spherical shape is cut out for each arm.

Multiple possible workspaces for the new parallel robot configuration are shown in Fig. 11. Figure 11 demonstrates the effects of varying the new link length and adjusting link d so the only difference between the workspace shown in Fig.10 and the new workspace is the length of link c. Thus, when link c=0 then the workspace is identical to the Delta robot Fig. 10.

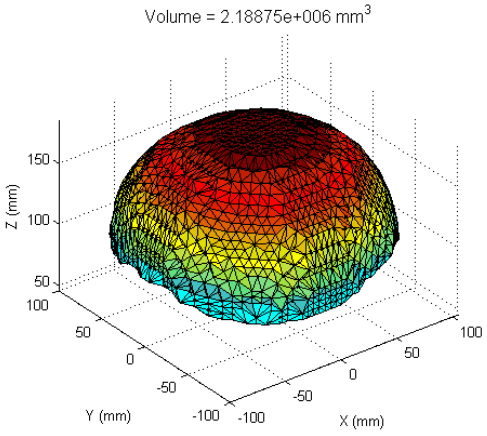
When the length of link c (labeled in Fig. 9) is varied from -40 to 40 while maintaining the distance between P_b and P_d (the length of link d=100+c) it is possible



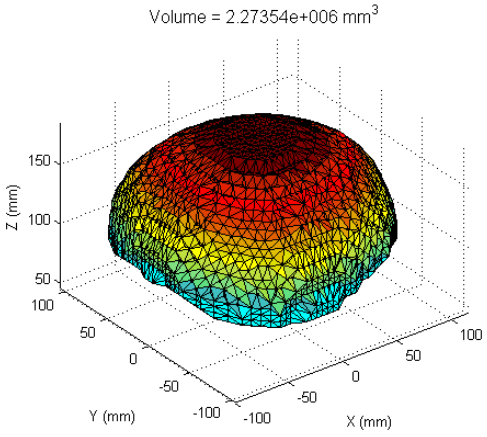
(a) Link c = -20



(b) Link c = -40



(c) Link c = +20



(d) link c = +40

Figure 11: Robot Workspace Where Link C is Varied

to graphically represent the effect of the new link on the workspace in Fig.11.

When the length of link c is negative (similar to the University of Maryland Manipulator in Fig. 4) it can be seen that the workspace is limited and in fact reduced. This is because the parallel linkage approaches a singularity quicker where the 4-bar linkage is horizontal. When the length of link c=40 and the length of link d=140 as seen in Fig.11d it can be seen that the workspace has a larger volume than the standard Delta robot. This illustrates the workspace improvement that the proposed link structure has over that of the standard Delta design.

4 Jacobian, Stiffness and Dexterity Analysis

4.1 Introduction

The Jacobian of a parallel robot mathematically relates the velocity of the end effector to the velocity of the actuated joints [14, 15]. For a parallel manipulator, the Jacobian is in general related to the Jacobian of a serial manipulator through its inverse.

The Jacobian can be used for more than calculating the joint speeds by performing a few more calculations it is possible to obtain the stiffness matrix which describes the Cartesian resistance of the robot to external forces at the end effector assuming the links are rigid and the actuated joints are compliant. The stiffness of the robot is a very important factor to consider optimizing as it affects both the maximum payload and the accuracy of the robot. Thus an optimization problem accounting for this can be essential to design.

Using the Jacobian matrix or even the stiffness matrix, it is then possible to calculate the dexterity of the robot, which is another important kinematic factor that can effect the pose accuracy of the parallel manipulator.

The labeling used in this chapter is given in Fig. 12.

The complete MATLAB code to calculate the stiffness and dexterity is given in Appendix B.

4.2 Jacobian

The Jacobian calculations require the link lengths used to calculate the inverse kinematics as well as the calculated angles for each actuated joint.

To calculate the Jacobian, the values of θ_1, θ_2 and θ_3 (θ for each arm in figure 9) are used from the inverse kinematics. The Jacobian is defined as $A\dot{x} = B\dot{q}$ where $\dot{x}$ is the Cartesian rate changes and $\dot{q}$ is the actuator rate changes. Then, $J = B^{-1}A$ as described in [15].

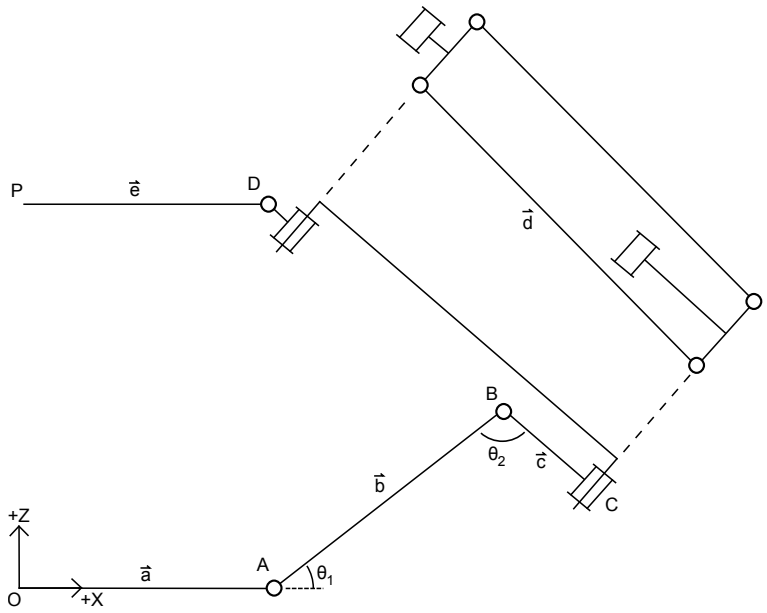


Figure 12: Schematic Labeling for Jacobian

The first step to calculate the Jacobian matrix is to take the inverse kinematics and take the derivative with respect to time. The inverse kinematics in closed loop vector form can be described as

$$\overrightarrow{OA_i} + \overrightarrow{A_iB_i} + \overrightarrow{B_iC_i} + \overrightarrow{C_iD_i} = \overrightarrow{OP} + \overrightarrow{PD_i} \tag{9}$$

Where i specifies a particular arm and $\overrightarrow{OA_i}$ is a vector from the origin to the first joint in an arm. $\overrightarrow{A_iB_i}$represents a vector from the first joint to the second joint in a given arm and so on. $\overrightarrow{OP}$ is the vector from the origin to the end effector and $\overrightarrow{PD_i}$ is the vector from the end effector the joint attached to the end effector for a given arm.

Taking the derivative with respect to time yields

$$\dot{\overrightarrow{A_iB_i}} + \dot{\overrightarrow{B_iC_i}} + \dot{\overrightarrow{C_iD_i}} = \dot{\overrightarrow{OP}} \tag{10}$$

which can be rearranged into

$$\omega_{1i} \times \hat{b_i} + \omega_{2i} \times \hat{c_i} + \omega_{2i} \times \hat{d_i} = \hat{p} \tag{11}$$

where ω_{1i} is the speed of the actuated joint, $\hat{b_i}$ is the vector of length b for link i and $\dot{p}$ is the velocity of the end effector.

Equation 11 can be rewritten in parametric form. This makes it possible to find a solution to the Jacobian equation by solving the three equations for the three unknowns.

The Jacobian was found by solving for A and B in the equation $A\dot{x} = B\dot{q}$ using Maple.

The Jacobian calculations were then converted into a form MATLAB can use, given the robot link lengths and joint angles for a given point in a workspace. In the calculations the Jacobian is defined as Jaa and $Jaa = Jq \setminus Jx$ which is equivalent to $Jaa = Jq^{-1}Jx = B^{-1}A$.

```
sin_theta11=sin(theta1(1)); % these come from the IK
sin_theta12=sin(theta1(2)); % these come from the IK
sin_theta13=sin(theta1(3)); % these come from the IK
cos_theta11=cos(theta1(1)); % these come from the IK
cos_theta12=cos(theta1(2)); % these come from the IK
cos_theta13=cos(theta1(3)); % these come from the IK
cos_theta31=(py*cos(phi1)-px*sin(phi1))/d;
cos_theta32=(py*cos(phi2)-px*sin(phi2))/d;
cos_theta33=(py*cos(phi3)-px*sin(phi3))/d;
sin_theta31=sqrt(1-cos_theta31^2);
sin_theta32=sqrt(1-cos_theta32^2);
sin_theta33=sqrt(1-cos_theta33^2);
sin_theta121=(pz-b*sin_theta11)/(d*sin_theta31-c);
```

```
sin_theta122=(pz-b*sin_theta12)/(d*sin_theta32-c);
sin_theta123=(pz-b*sin_theta13)/(d*sin_theta33-c);
cos_theta121=sqrt(1-sin_theta121^2);
cos_theta122=sqrt(1-sin_theta122^2);
cos_theta123=sqrt(1-sin_theta123^2);
% Jx - 3x3
Jx(1,1)=-cos_theta31*sin(phi1)+cos_theta121*sin_theta31*cos(phi1);
Jx(2,1)=-cos_theta32*sin(phi2)+cos_theta122*sin_theta32*cos(phi2);
Jx(3,1)=-cos_theta33*sin(phi3)+cos_theta123*sin_theta33*cos(phi3);
Jx(1,2)=cos_theta121*sin_theta31*sin(phi1)+cos_theta31*cos(phi1);
Jx(2,2)=cos_theta122*sin_theta32*sin(phi2)+cos_theta32*cos(phi2);
Jx(3,2)=cos_theta123*sin_theta33*sin(phi3)+cos_theta33*cos(phi3);
Jx(1,3)=sin_theta121*sin_theta31;
Jx(2,3)=sin_theta122*sin_theta32;
Jx(3,3)=sin_theta123*sin_theta33;
Jq=zeros(3,3);
Jq(1,1)=b*(sin_theta121*cos_theta11-sin_theta11*cos_theta121)
    *sin_theta31;
Jq(2,2)=b*(sin_theta122*cos_theta12-sin_theta12*cos_theta122)
    *sin_theta32;
Jq(3,3)=b*(sin_theta123*cos_theta13-sin_theta13*cos_theta123)
    *sin_theta33;
Jaa=Jq\Jx;
```

4.3 Stiffness Analysis

The stiffness of the parallel manipulator is defined as $Stm = k * J'_{aa} * J_{aa}$ [14, 16]. This method assumes that the links are rigid enough such that the majority of the compliance comes from the actuated joints. The instantaneous stiffness of the end effector in the x direction is given by Stm(1,1), y by Stm(2,2) and the z stiffness is Stm(3,3). The total stiffness is the trace of the stiffness matrix.

After some testing, it was observed that the z stiffness was only dependent on the z position within a given workspace, so to make the stiffness more effective for optimization, the final code only takes into account the x and y stiffness to amplify the variance within the workspace, but all the figures shown take the Z stiffness into account.

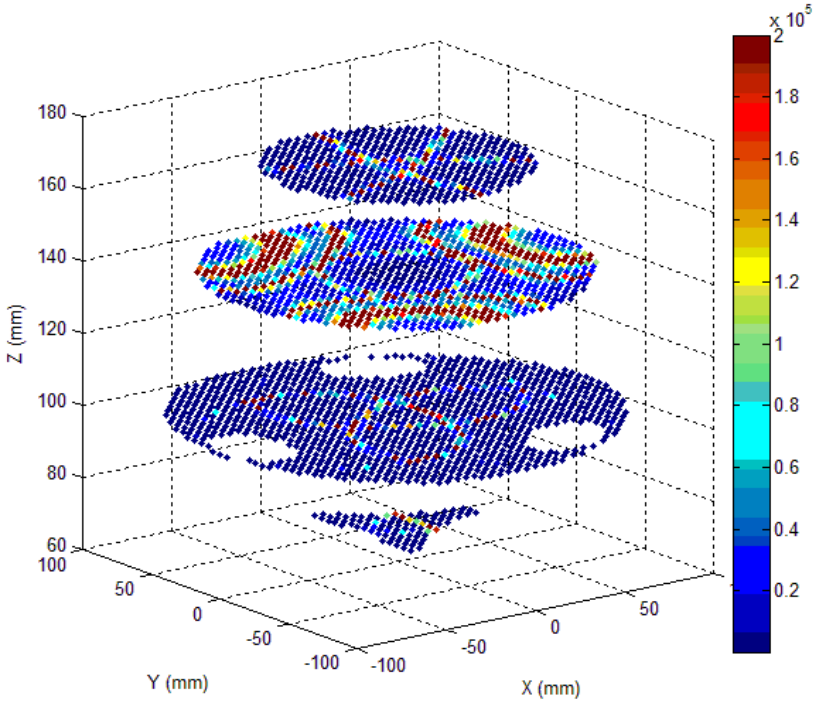
The average lateral stiffness of a workspace used in the optimization is calculated by averaging the x and y stiffness for each point in the reachable workspace.

Figure 13a shows the stiffness for the workspace where c=20 and d=120[1]. The sections are taken where z= 70,100,140, and 170. The dark blue points have a lower stiffness and as the colour changes to red the stiffness increases.

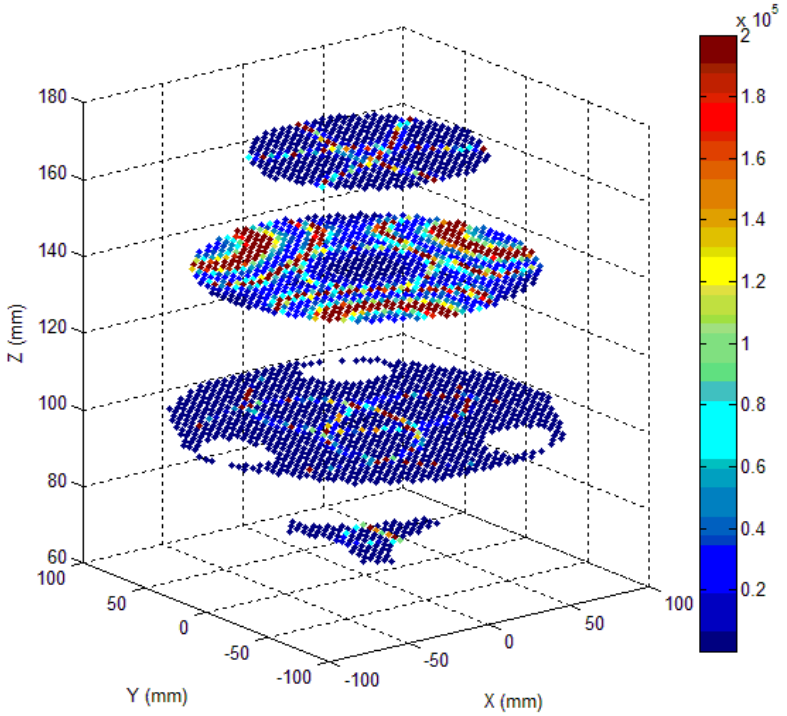
The colours in Fig. 13 have been adjusted by limiting the stiffness to a maximum of 200000. This allows the more subtle features to show better, otherwise only a few points with a very high stiffness can be seen. Overall, only a few points are substantially affected by this imposed limit. The lowest value points have a value of around 100, so there are no points where the robot is unusable due to stiffness.

It can be seen in Fig. 13a that the stiffness is relatively constant across the workspace, which is consistent with previous results[8]. Because Fig. 13 shows lateral stiffness there are a few area with higher stiffness as the manipulator approaches a singularity. These areas don't adversely affect the stiffness, in fact they are good because the stiffness is higher.

[1]See Fig. 9

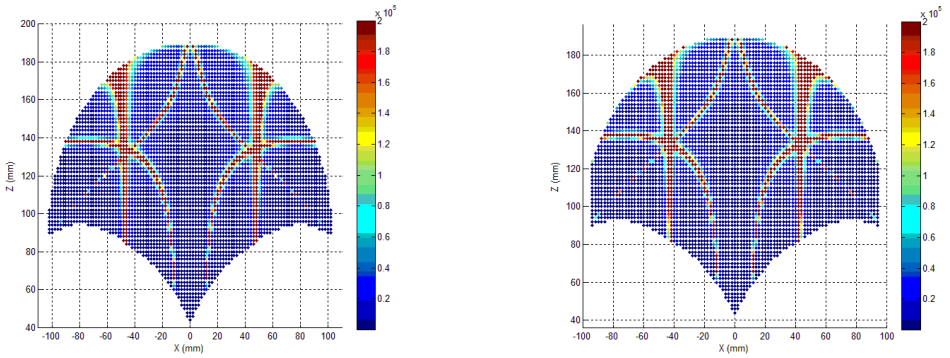


(a) c=20 d=120



(b) c=0 d=100

Figure 13: Slices of Workspace Showing Average Stiffness



(a) c=2 d=120 (b) c=0 d=100

Figure 14: Stiffness Slices Along X-Z Plane

To see the effect of the new link length on the stiffness Fig. 13b shows the same stiffness slices where the length of link c=0 and d=100, effectively removing the new link so it reverts back to a Delta robot. By comparing Fig. 13a and Fig. 13b it can be seen that the stiffness is not significantly affected by link c. At first this may seem counter-intuitive, but the position of the 4-bar linkage, which is what c mainly affects, does not affect the stiffness.

For additional comparison, slices in the X-Z plane for each link length configuration are included in Fig. 14.

4.4 Dexterity Analysis

The dexterity of a robot can be measured by the condition number of the Jacobian matrix. Which is written as $dexterity = \|J^{-1}\| \, \|J\|$ [17, 18, 19]. The dexterity of the parallel robot is also defined in the equations below as it relates to the stiffness matrix, Stm.

$$\lambda_1 = max(eig(Stm))$$

$$\lambda_2 = min(eig(Stm))$$

$$dexterity = \sqrt{\frac{\lambda_1}{\lambda_2}}$$

By using the min and max Eigenvalues of the stiffness matrix, we can avoid having to calculate the inverse of the Jacobian, which saves valuable time during optimization. Since the Eigenvalues are used as a ratio, the effect of k (scalar value) from the stiffness matrix is also negated.

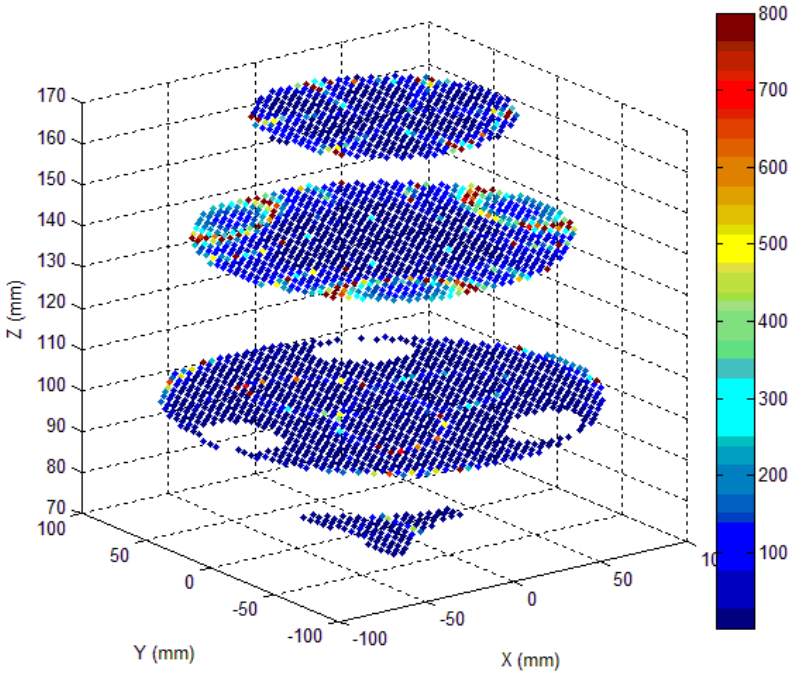
Like the stiffness, the dexterity for all reachable points in a workspace are averaged to get a single dexterity value for use when optimizing the workspace.

Figure 15a shows the dexterity for the workspace where c=20 and d=120. The sections are taken where z= 70,100,140, and 170. The dark blue points have a lower dexterity value, which is further from a singularity, and as the colour changes to red the dexterity becomes worse.

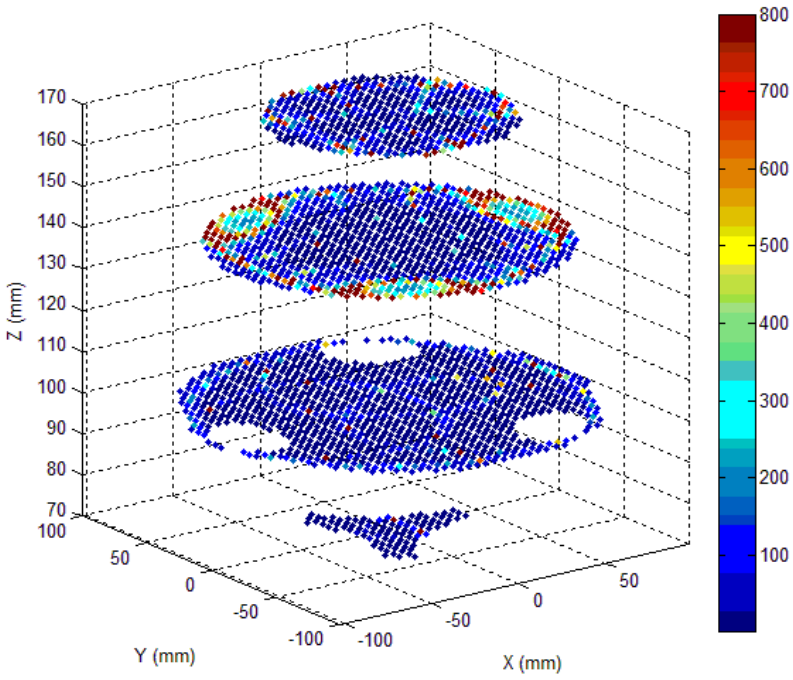
The colours in Fig. 15 have been adjusted by limiting the dexterity to a maximum of 800. This allows the more subtle features to show better, otherwise only a few points with high values can be seen in the colouring.

By comparing Fig. 15a and 15b it is possible to see the effect of the new link on the dexterity of the robot, similar to how the stiffness was analyzed. This preliminary comparison shows that link c has a major effect on the workspace. Not only is there less red in the images, the red that there is is further from the center of the workspace. This means that the new link effectively pushes the singularities away from the workspace, making it more useable.

For additional comparison, slices in the X-Z plane for each link length configuration are included in Fig. 16.

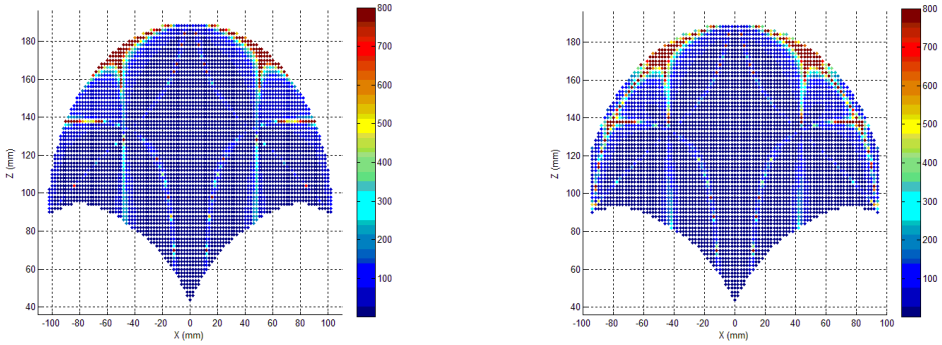


(a) c=20 d=120



(b) c=0 d=100

Figure 15: Slices of Workspace Showing Dexterity



(a) c=2 d=120 (b) c=0 d=100

Figure 16: Dexterity Slices Along X-Z Plane

5 Optimization

5.1 Introduction

Many people have undertaken various methods to calculate how to make the workspace more useable [10, 20, 21, 22]. The most common aspect of a parallel manipulator chosen to optimize is the workspace volume.

The new link length[2] as well as the length of the parallel links[3] are optimized using the built-in MATLAB optimization toolbox. Because there are two parameters to optimize, and the volume affects both linearly, more parameters are needed to fully constrain the problem into a true multi-objective optimization problem. The two other parameters selected are the stiffness and dexterity.

5.2 Optimization Setup

Each of the calculated aspects of the workspace; volume, stiffness and dexterity, have opposing effects on the design parameters selected for optimization. This was verified by manually varying one design parameter at a time and observing the trend in the three output values. By setting the objective function result to be a combination

[2]Length c in figure 9
[3]Length d in figure 9

Table 2: Volume, Stiffness, and Dexterity Results by Manually Varying Inputs

(a) Varying $d - c = 200$

Link c length	-20.0	-10.0	0.0	10.0	20.0	30.0
Link d length	180.0	190.0	200.0	210.0	220.0	230.0
Volume (mm^3)	4970000	5308800	5633300	5713500	5770500	5826300
Avg. Stiffness	2289.0	2317.7	2355.7	2340.1	2325.3	2314.2
Avg. Dexterity	97.0625	92.2544	89.0287	81.9184	74.2435	64.7178

(b) Varying length d

Link c length	20.0	20.0	20.0	20.0	20.0	20.0
Link d length	100.0	160.0	180.0	200.0	220.0	240.0
Volume (mm^3)	1247000	3525700	4226400	4968800	5770500	6632900
Avg. Stiffness	2800.9	3263.2	2818.6	2524.6	2325.3	2183.1
Avg. Dexterity	49.5529	65.3703	67.1334	70.1852	74.2435	78.4530

of each of the individual objectives (stiffness, dexterity, workspace) an optimization problem is formulated.

To explore how the different link lengths affect the volume, average stiffness, and average dexterity the link lengths were varied and the results recorded into tables. The average stiffness and dexterity are found by calculating the stiffness and dexterity at each point within a workspace and then averaging all the points.

Table 2a shows the results by keeping $d - c = 100$; this better allows the effects of the new link to be seen. When $c < 0$ it drastically reduces the workspace, slightly reduces the stiffness, and increases the dexterity[4]. But when $c > 0$ it positively affects almost all the parameters; the volume is slightly increased, the dexterity is reduced and the stiffness is only slightly reduced. From these results it can be seen that link c should optimize to a positive number without going to the upper bound because the stiffness constrains it so it will not get too long. Without even optimizing the structure, these results hint at the benefits of the new link directly.

Table 2b shows the affect of varying link d, the length of the 4-bar linkage, on the volume, stiffness, and dexterity. It can be seen that increasing link d increases the

[4]meaning the singularities are closer to the workspace

volume, slightly reduces the stiffness and decreases the dexterity. These two tables confirm that the volume, average lateral stiffness and average dexterity make good candidates to optimize link c and d lengths because there are balanced parameters on both links.

In MATLAB, the optimization method used is simulated annealing. It requires an objective function that takes, in the case of this parallel robot, 2 inputs and returns a single value to optimize. The objective function calculates the three parameters and multiplies them by selected weights and then adds them up to get one value to optimize.

The optimization requires a point to start searching from, which was chosen as [20,200]. Bounds are also required by the method, and were selected as -50<c<100 and 50<d<300.

The function objF uses 2 inputs, the first one is the length of link c in Fig. 9 and the second input is length d in the same Fig. The weights are selected by a process of trial and error in order to get a balanced result that didn't optimize to a bound. If different parameters are required for an application appropriate weights can be selected.

The objective function used in the optimization problem is:

$$y = -0.0000019 * workspaceVolume - 0.0024 * averageStiffness + 0.000015 * averageDexterity \tag{12}$$

This formula sets up all of the parameters so that when it is minimized the parameters will be increased or decreased as needed(maximize volume and stiffness and minimize dexterity). The weights used in the output formula are selected by a process of trial and error to produce optimal results.

If improper weights are selected for optimization, then the workspace optimization will fail or go to the set bounds. An example is when the stiffness is too low then the

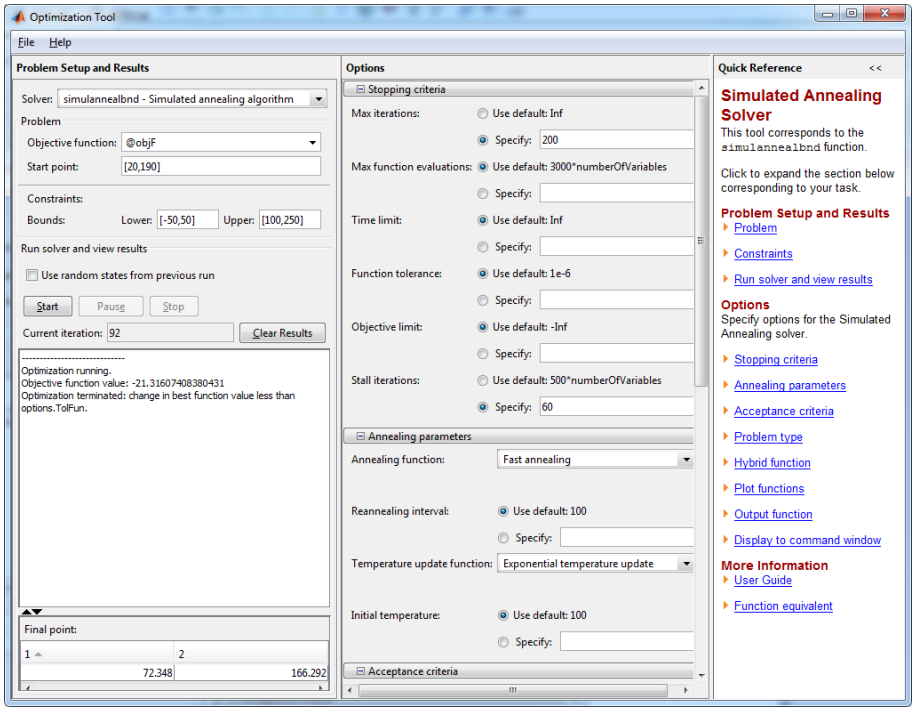


Figure 17: MATLAB Optimization Toolbox

length of link c will be negative and the length of link d will go to the upper bound. Thus, weights that try and provide opportunity for each sub-objective were used.

An example of the toolbox setup used is shown in figure 17.

The code for the objective function is given in Appendix C and requires the code in Appendix A and Appendix B to work.

5.3 Optimization Results

An example of the function value being optimized is shown in figure 18. The graph plots the current function value, but as seen in the image it fluctuates up and down. This is because of the method used to optimize; when a local minimum is found the algorithm continues to search for other local minima. When it cannot find any better values the lowest function value from all of the configurations is selected as the optimal value.

The input x values that the optimizer used are shown in Fig.19. It can be seen

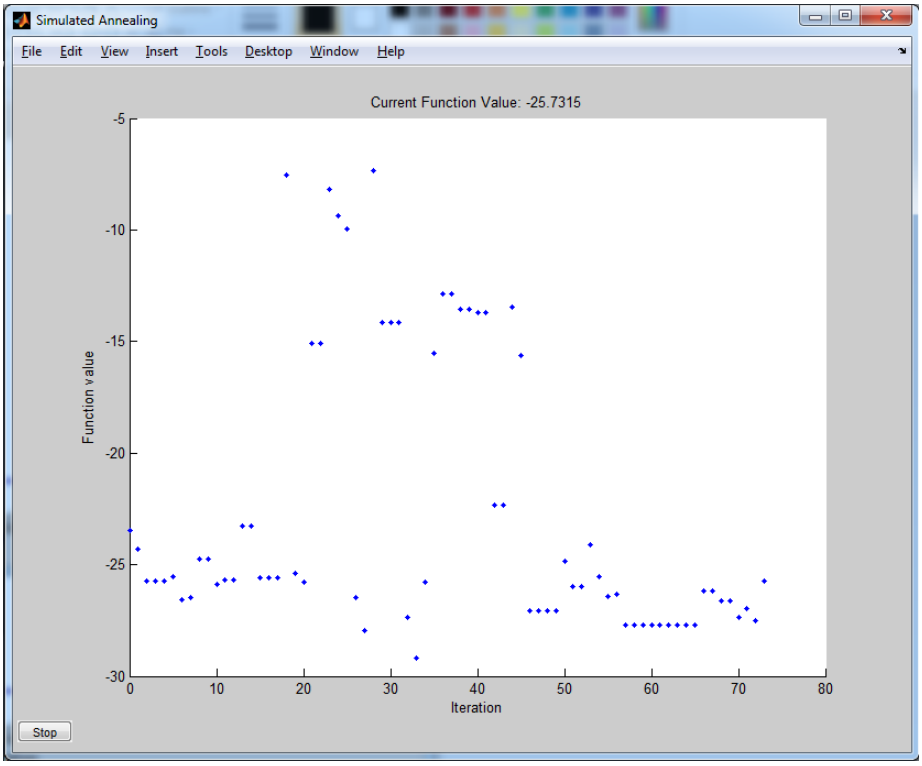


Figure 18: Optimization Function Value Vs Iteration

that the inputs vary greatly but they slowly narrow in on the optimal values.

The optimized results are a length of 72.348 mm for link c and 166.292 mm for link d. Figure 20 shows the final workspace plotted from a top view and Fig. 21 shows the underside of the same workspace.

It can be seen in Fig. 20 and 21 that there are very few extraneous geometric features[5], meaning that most of the workspace is useable.

Slices of the stiffness and dexterity of the optimized robot are shown in Fig. 22.

Like in the unoptimized workspace, the stiffness shows little variance apart from a slight stiffness increase nearer to some singularities. Unlike the unoptimized workspace, the dexterity is more evenly distributed and overall has a lower average, making for a better workspace, that is more isotropic in terms of its properties.

By setting the length of link c[6] to 0 while keeping the distance between P_b and

[5]The exception being the point on the bottom of the workspace.
[6]See Fig. 9

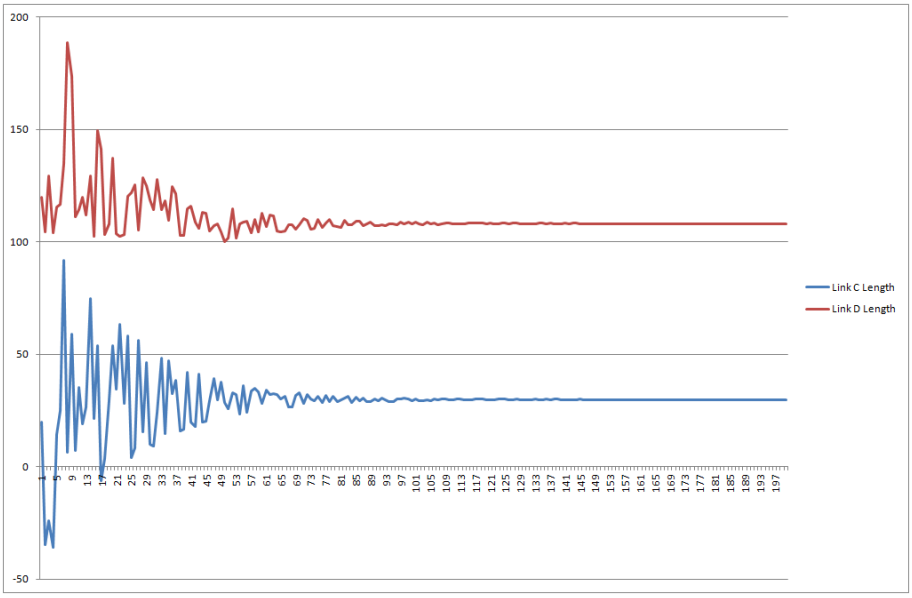


Figure 19: Optimization Function Inputs to the Objective Function Over Time

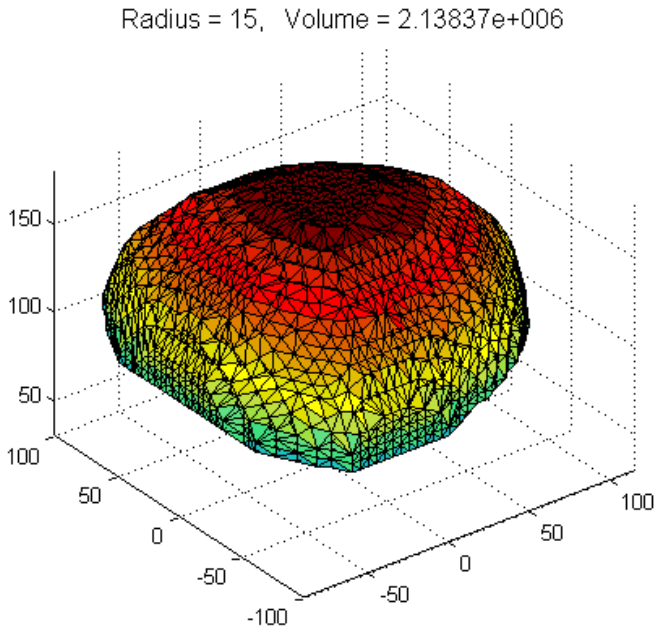


Figure 20: Optimized Workspace (Top Trimetric View)

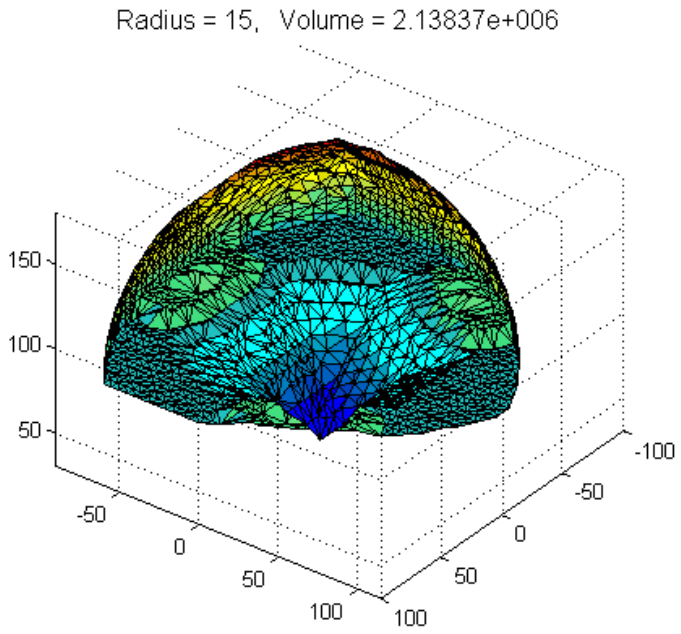
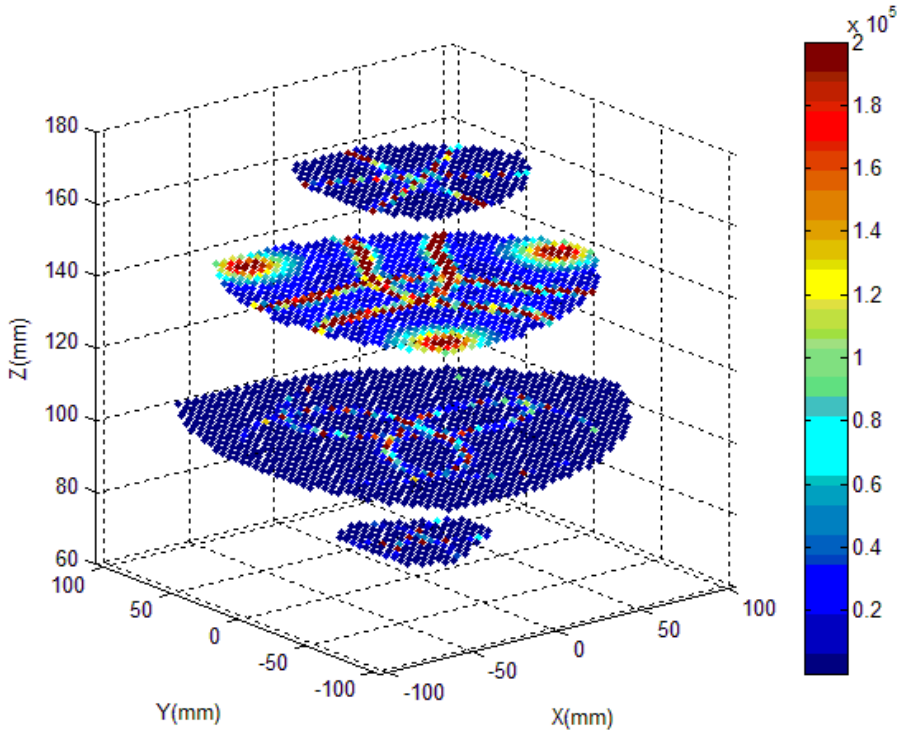


Figure 21: Optimized Workspace (Bottom Trimetric View)

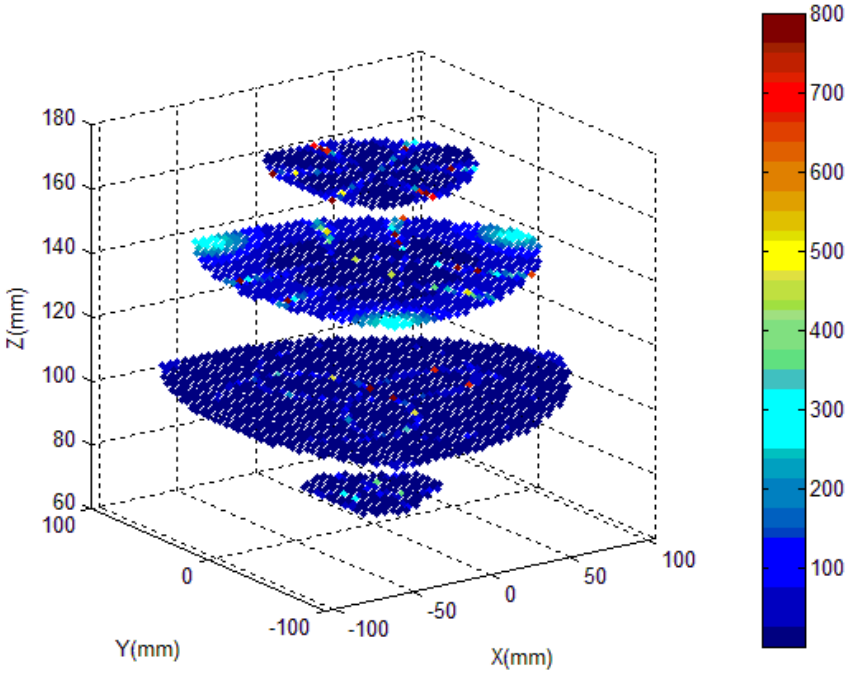
P_d constant[7], it is possible to compare the effect of the new link on the workspace.

The volume of the workspace with link c is $2098100mm^3$ and the volume where link c=0 is $1778200mm^3$; the addition of link c increases the workspace volume by 17.7%. The change in the average stiffness is minimally worsened by 4.0% and the dexterity is improved by 53.1%. The new link moves the singularity where link d is horizontal further away from the workspace, making it more useful as shown by the dexterity improvement in the optimized workspace.

[7] c=0 and d=94



(a) Optimized stiffness



(b) Optimized Dexterity

Figure 22: Optimized Stiffness and Dexterity

6 Prototype

Sometimes when designing a robot from a purely mathematical perspective, it is easy to overlook some key design flaws that make the robot unusable. For this reason it is necessary to design and build a prototype to verify the design and iron out any problems.

The prototype described in this chapter was designed in UGS NX7.5 and created using a laser cutter for the acrylic parts and a 3D printer for the parts made from PLA.

6.1 Design

The prototype design shown in Fig. 23 is designed to be as environmentally friendly as possible while still using use off-the-shelf parts. To achieve this, no machined axles were used, instead the screws function to hold it together as well as act as joints where necessary. Some plastic parts are also printed in PLA, a biodegradable plastic made from corn.

To aid in manufacturing, most of the parts are designed to be 2-1/2 dimensional, meaning the outline can be sketched in 2 dimensions and then it is extruded to a 3 dimensional object. To allow this type of part to create 3 dimensional assemblies these parts are assembled using a t-slot style connection. One part has tabs that fit into holes of another and the two are held at a right angle using a nut and bolt.

Where a joint is required a bolt and lock nut are used. This allows the bolt to fit snugly, but not so much as to restrict any possible rotation about its axis.

All bolts and nuts are intentionally left out of the CAD pictures to simplify the look in Figs. 23 and 24.

Figure 24 shows an adjust position for the end effector, demonstrating how the links move.

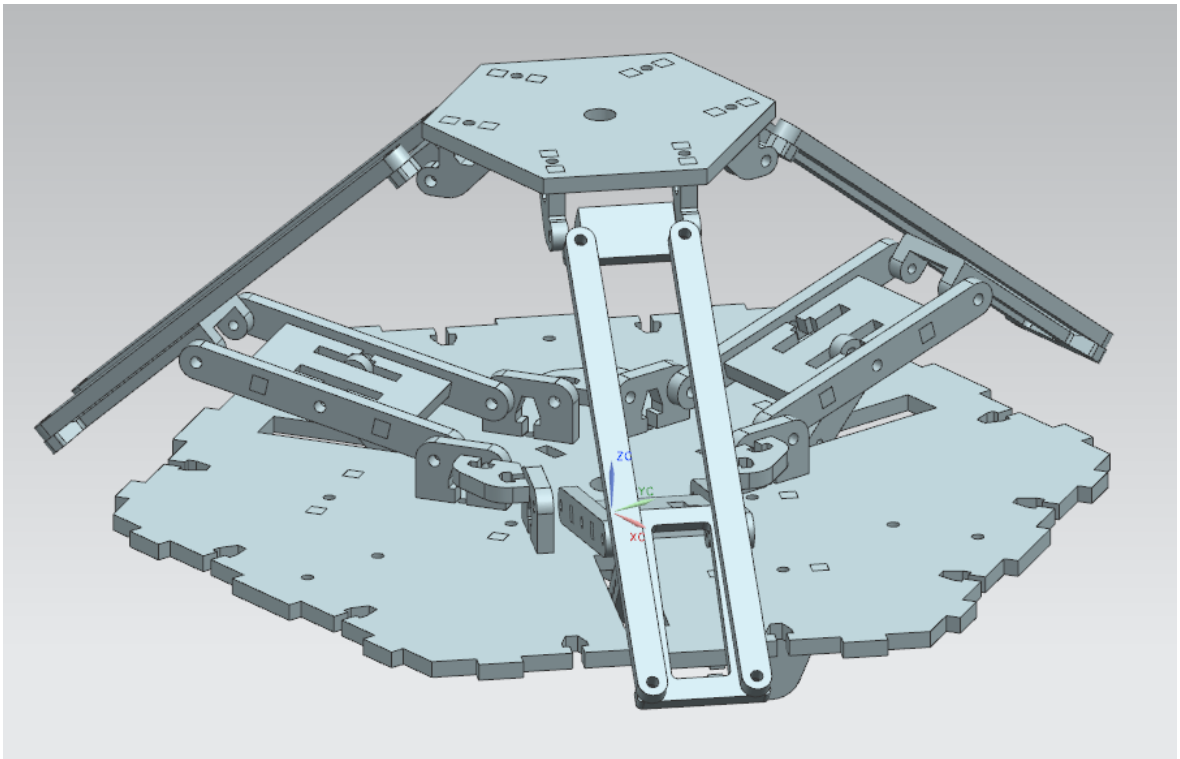


Figure 23: NX7.5 CAD Model

6.2 Constructed Prototype

The prototype fabricated from the CAD model is shown in figure 25.

The prototype is controlled by 3 hobby servo motors controlled by a ATMEGA328P microcontroller programmed using C++. The code to move the robot through a series of patterns is given in Appendix D.

Two alternate positions of the prototype at the edge of its workspace can be seen in Fig. 26. Figure 26a shows the prototype as it approaches a singularity where the links are all vertical and Fig. 26b shows the prototype as it approaches the horizontal limit. Two of the 4 bar linkages can be seen approaching the singularity where they are completely horizontal and the leftmost link is at its extent because $\theta = 0$.

The prototype was tested by programming it to run through various patterns, such as a helical spiral and squares at different heights. By observing the movements it confirms the functionality.

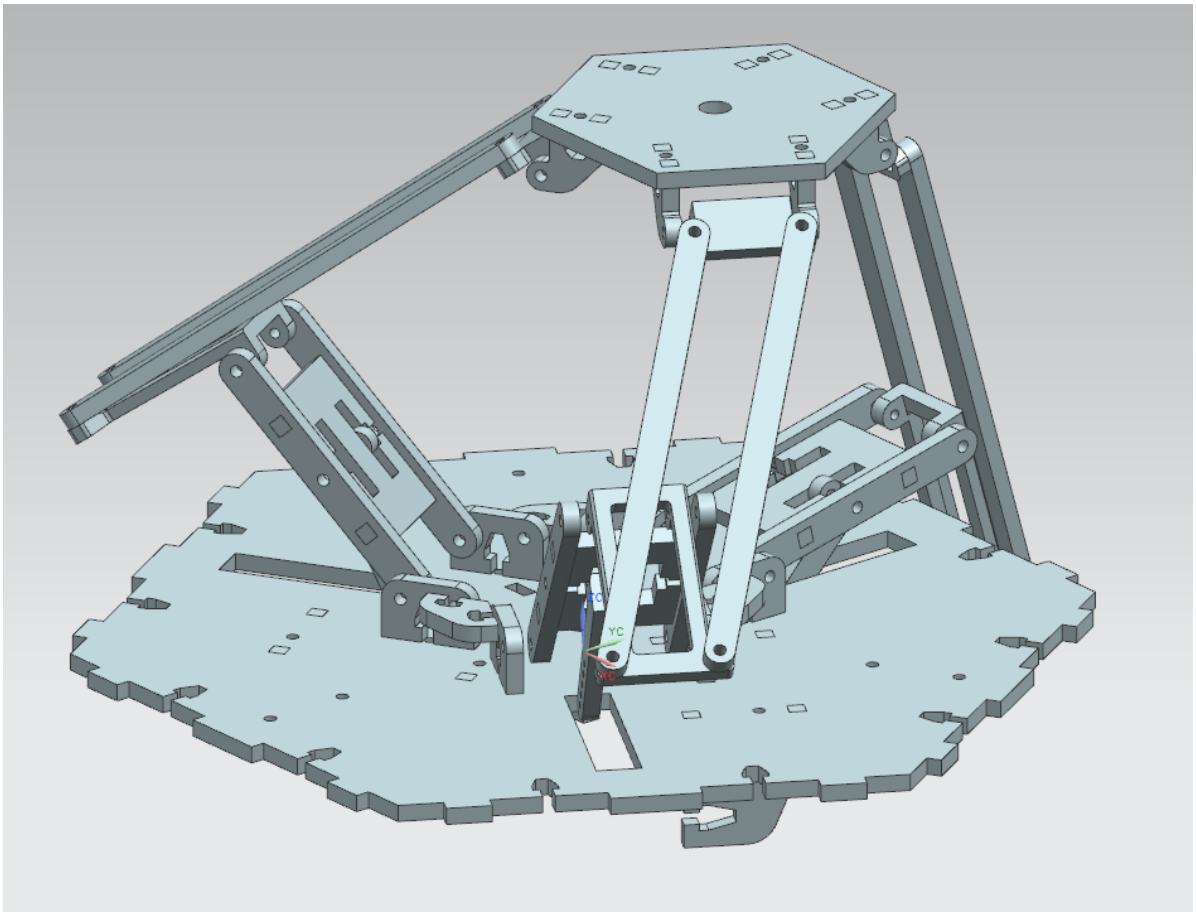


Figure 24: NX7.5 CAD Model Adjusted

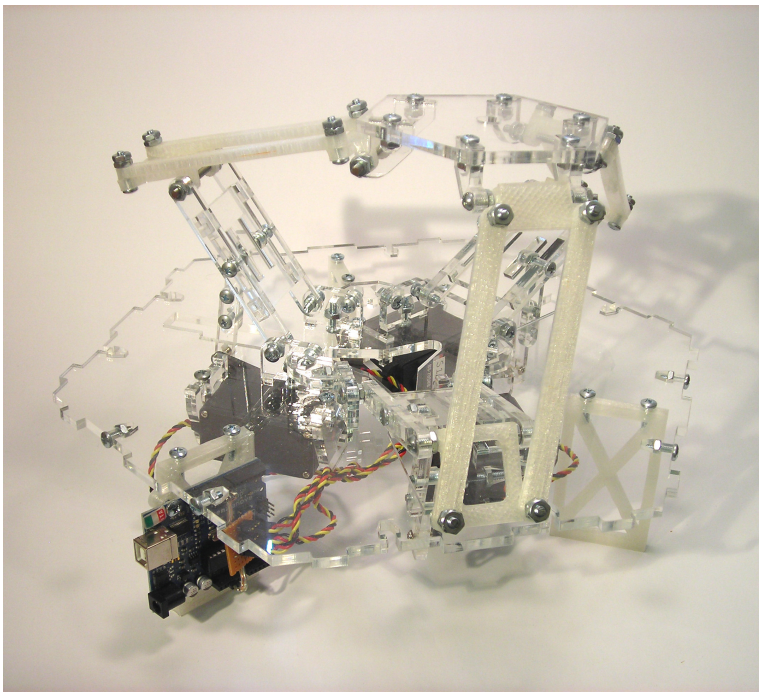
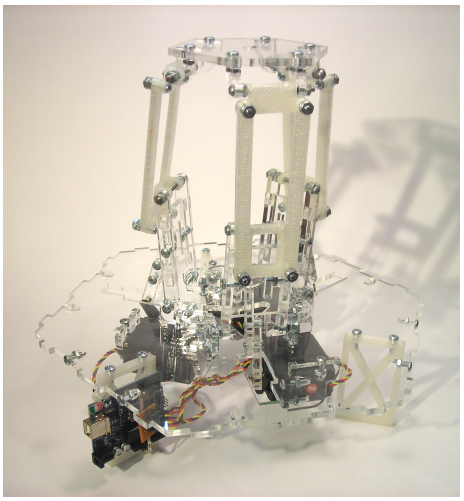
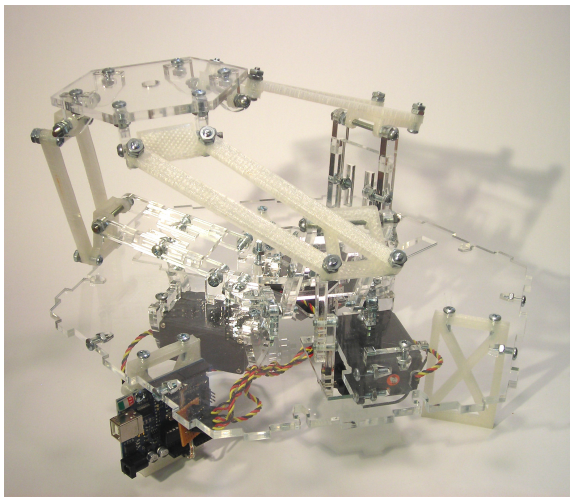


Figure 25: Prototype



(a) Prototype Maximum Height



(b) Prototype Max Horizontal

Figure 26: Prototype Alternate Positions

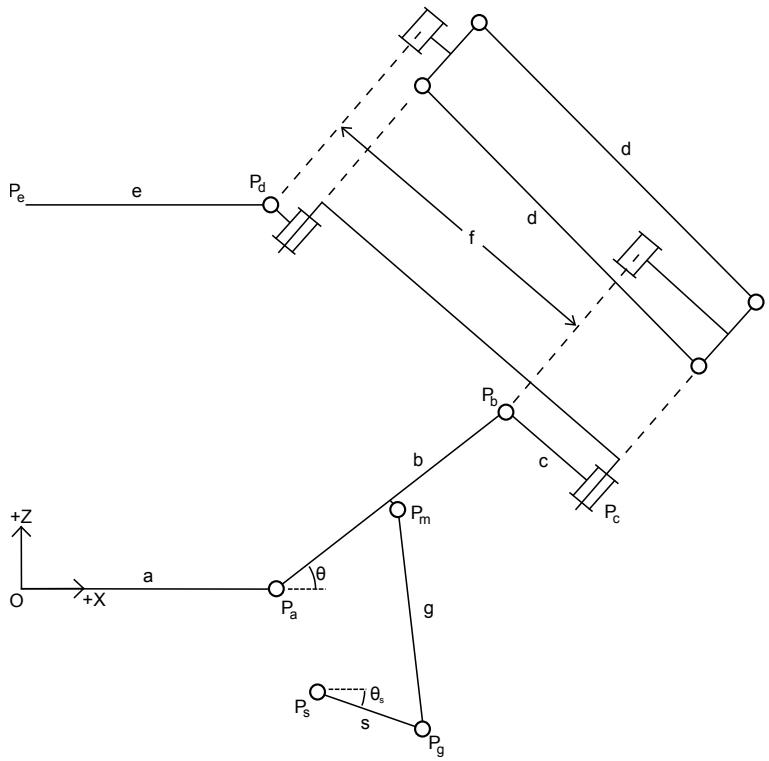


Figure 27: New Single Arm Schematic with additional linkage

The constructed prototype verifies the calculations and proves that this evolution of the Delta robot is valid and useful.

7 Additional Features and Future Work

An addition to the robot is shown in Fig.27, it uses a 4-bar linkage in the X-Z plane of each arm to reduce the effect of the movement of the actuator on link b. This allows the same motor to have a higher torque and, if designed properly, it can prevent any damage to the robot caused by over extension of a joint. This link can be seen in the prototype in Fig. 25 connected to a hobby servo. Link s in each arm is the horn of the hobby servo.

Future analysis of the prototype could be done by measuring the actual position of the end effector and comparing it to the theoretical position. It would then be possible to plot the workspace coloured to show error. This could then be used to better design

future versions and avoid the areas of the workspace with greater positional error.

8 Summary

In conclusion, this thesis has presented a new variant of a Delta robot with distinct advantages. The new link not only increases the volume of the workspace but also the stiffness of the existing workspace. The work first covers the inverse kinematics and how the workspace is affected by the new link. It continues by calculating the Jacobian, stiffness and dexterity; which culminates in using MATLAB to optimize two link lengths.

To further verify functionality of the new robot, a prototype was designed, constructed and programmed to move through a series of patterns. The prototype works as expected and reinforces the findings.

A novel variant of the Delta parallel robot has been presented with a direction for continued work to improve the design.

References

[1] VE Gough. Contribution to discussion of papers on research in automotive stability, control and tyre performance. In *Proc. Auto Div. Inst. Mechanical Engineers*, volume 1957, pages 371–386, 1956.

[2] D. Stewart. A platform with six degrees of freedom. *Proceedings of the Institution of Mechanical Engineers*, 180(1):371–386, 1965.

[3] F. Gao, W. Li, X. Zhao, Z. Jin, and H. Zhao. New kinematic structures for 2-, 3-, 4-, and 5-dof parallel manipulator designs. *Mechanism and Machine Theory*, 37(11):1395 – 1411, 2002.

[4] L.W. Tsai and R.E. Stamper. A parallel manipulator with only translational degrees of freedom. 1997.

[5] R. Clavel. *Conception d'un robot parallèle rapide à 4 degrés de liberté.* PhD thesis, Lausanne, 1991.

[6] F. Pierrot, C. Reynaud, and A. Fournier. Delta: a simple and efficient parallel robot. *Robotica*, 8(02):105–109, 1990.

[7] V. Nabat, M. de la O Rodriguez, O. Company, S. Krut, and F. Pierrot. Par4: very high speed parallel robot for pick-and-place. In *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 553 – 558, aug. 2005.

[8] R.E. Stamper. *A three degree of freedom parallel manipulator with only translational degrees of freedom.* PhD thesis, 1997.

[9] D.C. Carp-Ciocardia and S. Staicu. Dynamics of delta parallel robot with prismatic actuators. In *Mechatronics, 2005. ICM '05. IEEE International Conference on*, pages 870 –875, july 2005.

[10] Z. Wang, S. Ji, Y. Li, and Y. Wan. A unified algorithm to determine the reachable and dexterous workspace of parallel manipulators. *Robotics and Computer-Integrated Manufacturing*, 26(5):454 – 460, 2010.

[11] Y.W. Li, J.S. Wang, L.P. Wang, and X.J. Liu. Inverse dynamics and simulation of a 3-dof spatial parallel manipulator. In *Robotics and Automation, 2003. Proceedings. ICRA'03. IEEE International Conference on*, volume 3, pages 4092–4097. IEEE, 2003.

[12] X. Liu and J. Wang. Some new parallel mechanisms containing the planar four-bar parallelogram. *The International Journal of Robotics Research*, 22(9):717–732, 2003.

[13] F. Pierrot, A. Fournier, and P. Dauchex. Towards a fully-parallel 6 dof robot for high-speed applications. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1288 –1293 vol.2, apr 1991.

[14] D. Zhang. *Parallel Robotic Machine Tools*. Springer, New York,NY, 2010.

[15] J. P. Merlet. Jacobian, manipulability, condition number, and accuracy of parallel robots. *Journal of Mechanical Design*, 128(1):199–206, 2006.

[16] C. Gosselin. Stiffness mapping for parallel manipulators. *Robotics and Automation, IEEE Transactions on*, 6(3):377–382, 1990.

[17] J. Angeles. *Fundamentals of Robotic Mechanical Systems*. Springer, New York,NY, 2007.

[18] M. Arsenault and R. Boudreau. Synthesis of planar parallel mechanisms while considering workspace, dexterity, stiffness and singularity avoidance. *Journal of Mechanical Design*, 128(1):69–78, 2006.

[19] T. Huang, D.J. Whitehouse, and J. Wang. The local dexterity, optimal architecture and design criteria of parallel machine tools. *CIRP Annals - Manufacturing Technology*, 47(1):346 – 350, 1998.

[20] M.A. Laribi, L. Romdhane, and S. Zeghloul. Analysis and dimensional synthesis of the delta robot for a prescribed workspace. *Mechanism and Machine Theory*, 42(7):859 – 870, 2007.

[21] J.A. Carretero, R.P. Podhorodeski, M.A. Nahon, and C.M. Gosselin. Kinematic analysis and optimization of a new three degree-of-freedom spatial parallel manipulator. *Journal of Mechanical Design*, 122(1):17–24, 2000.

[22] P. Vischer and R. Clavel. Kinematic calibration of the parallel delta robot. *Robotica*, 16:207 – 218, 1998.

Appendix A

This code calculates the lower link angles given the position of the end effector

The code needs to be enclosed in a file called deltaInverse.m

```
function [theta,thetaServo,errorFlag] =

    deltaInverse(x,y,z,linkLengths)

  if(nargin<4)

     linkLengths=[50,90,15,150,50.65,70,28.2];

  end

  %trig constants used to rotate coordinate system

  cos120=cos(120*pi/180);

  sin120=sin(120*pi/180);

  errorStatus=0;

  %make sure the suggested geometry is valid

  if(linkLengths(4)-linkLengths(3)<20)

    errorStatus=-0.5;

  end

  % rotate coordinates to +120 deg and calculate second arm

  if (errorStatus == 0)

    [theta1 thetaServo1 errorStatus]=

        deltaSingleArm(x,y,z,linkLengths);

  end

  % rotate coordinates to +120 deg and calculate second arm

  if (errorStatus == 0)

    [theta2 thetaServo2 errorStatus]=

     deltaSingleArm(x*cos120 + y*sin120,
```

```
        y*cos120-x*sin120, z, linkLengths);
    end
    % rotate coordinates to -120 deg and calculate third arm
    if (errorStatus == 0)
      [theta3 thetaServo3 errorStatus]=
        deltaSingleArm(x*cos120 - y*sin120,
          y*cos120+x*sin120, z, linkLengths);
    end
    %update the output only if everything works
    if (errorStatus == 0)
      errorFlag=0;
      theta=[theta1,theta2,theta3];
      thetaServo=[thetaServo1,thetaServo2,thetaServo3];
    else
      errorFlag=errorStatus;
      theta=[0,0,0];
      thetaServo=[0,0,0];
    end
  end
  function [thetaArm,thetaServo,errorAlert]=
  deltaSingleArm(x0,y0,z0,linkLengths)
    if(nargin<4)  % mechanism dimensions (general constants) in mm
      %'default link lengths assumed'
      a=50;       %distance from origin to first r joint
      b=90;       %length of first link
      c=20;       %new length, +ve is extra parallel length
```

```
  d=100+c;    %length of parallel linkage
  e=50.65;    %length from end point to r joint on end effector
  %length of the link that connects the servo to the first
  %delta link
  g=70;
  s=28.2;         %length of the servo horn
else
  a=linkLengths(1);
  b=linkLengths(2);
  c=linkLengths(3);
  d=linkLengths(4);
  e=linkLengths(5);
  g=linkLengths(6);
  s=linkLengths(7);
end   Ps=[63,0,-40]; %position of the servo joint
Pa=[a,0,0];    %position of the r joint on the base

% Variables used during calculations
f=0;%length between end of first link and r joint on end effector
Pee=[x0,y0,z0];%position of the end effector
Pd=[e,0,50]; %position of the r joint on the end effector
Pb=[0,0,0];  %position where the two links meet
Pg=[0,0,0];  %point on the end of the servo horn
Pm=[0,0,0];  %point where the servo link meets the base delta arm
% Physical joint limits
maxDeltaArmAngle=90*pi/180;
minDeltaArmAngle=0*pi/180;
```

```
maxServoAngle=(90.0-1.0)*pi/180;%max angle from center
minServoAngle=-(90.0-1.0)*pi/180;%min angle from center
Pa(1)=a;
Pa(2)=0;
Pa(3)=0;
Pd(1)=x0+e;
Pd(2)=y0;
Pd(3)=z0;%position of the r joint on the end effector
%check if f is a real distance and not horizontal
if(abs(Pd(2))>(d*0.95))
  errorAlert=-1;
  thetaArm=0;
  thetaServo=0;
  return
end
%calculate projected length of parallel link
f=sqrt(d^2-Pd(2)^2)-c;
%calculate distance between Pa and Pd
dCalc=sqrt((Pd(1)-Pa(1))^2+(Pd(3)-Pa(3))^2);
%if it is an invalid position
if(dCalc>b+f||dCalc<abs(f-b)||dCalc==0)
  errorAlert=-2;%return error 1
  thetaArm=0;
  thetaServo=0;
  return
end
aCalc=(b^2-f^2+dCalc^2)/(2*dCalc);  %see notes
```

```
hCalc=sqrt(b^2-aCalc^2);            %see notes
%calculate x for where 2 links meet
Pb(1)=Pa(1)+aCalc*(Pd(1)-Pa(1))/dCalc+hCalc*(Pd(3)-Pa(3))/dCalc;
%calculate y for where 2 links meet
Pb(2)=0;
%calculate z for where 2 links meet
Pb(3)=Pa(3)+aCalc*(Pd(3)-Pa(3))/dCalc+hCalc*(Pd(1)-Pa(1))/dCalc;
if Pb(3)>Pd(3)
  errorAlert=-3;
  thetaArm=0;
  thetaServo=0;
  return
end
%angle of the delta robot arm
theta1=atan2(Pb(3)-Pa(3),Pb(1)-Pa(1));
if theta1<minDeltaArmAngle||theta1>maxDeltaArmAngle
  errorAlert = -4;%return error 4
  thetaArm=0;
  thetaServo=0;
  return
end
%extra servo link calculations begin here (same method as above,
%different values)
Pm(1)=(Pb(1)-Pa(1))/2+Pa(1);
Pm(2)=0;
Pm(3)=(Pb(3)-Pa(3))/2+Pa(3);
dCalc=sqrt((Pm(1)-Ps(1))^2+(Pm(3)-Ps(3))^2);
```

```
  %if it is an invalid position
  if dCalc>g+s||dCalc<abs(g-s)||dCalc==0
    errorAlert = -5;%return error 5
    thetaArm=0;
    thetaServo=0;
    return
  end
  aCalc=(s*s-g*g+dCalc*dCalc)/(2*dCalc);%see notes
  hCalc=sqrt(s*s-aCalc*aCalc);%see notes
  %calculate x for where servo horn meets link
  %calculate y for where servo horn meets link
Pg(1)=(Ps(1)+aCalc*(Pm(1)-Ps(1))/dCalc)+(hCalc*(Pm(3)-Ps(3))/dCalc);
  %calculate z for where servo horn meets link
  Pg(2)=0;
  %make sure z is valid with check below (2 possible solutions)
Pg(3)=(Ps(3)+aCalc*(Pm(3)-Ps(3))/dCalc)-(hCalc*(Pm(1)-Ps(1))/dCalc);
  temp1=sqrt((Pg(1)-Ps(1))^2+(Pg(3)-Ps(3))^2);
  calcTolerance=0.1;
  %change it if the code guessed wrong
 if temp1>s+calcTolerance||temp1<s-calcTolerance
Pg(3)=(Ps(3)+aCalc*(Pm(3)-Ps(3))/dCalc)+(hCalc*(Pm(1)-Ps(1))/dCalc);
    %calculate z for where servo horn meets link
  end

  theta2=atan2(Pg(3)-Ps(3),Pg(1)-Ps(1));%calculate servo theta
  if theta2<minServoAngle||theta2>maxServoAngle
    errorAlert = -6;%return error 6
```

```
        thetaArm=0;
        thetaServo=0;
        return
      end
      errorAlert = 0;
      thetaArm=theta1;%return the delta arm angle
      thetaServo=theta2;%returns the servo angle
    end
```

Appendix B

This code calculates the stiffness and dexterity of the robot for the given position. It requires the link angles calculated in the inverse kinematics and the robot link lengths.

The code needs to be enclosed in a file called deltaStiffnessAndDexterity.m

```
function [stiffness,dexterity,errorFlag]=


    deltaStiffnessAndDexterity(px,py,pz,theta1,linkLengths)


  phi1=0; phi2=120*pi/180; phi3=240*pi/180;%angle between arms
  if(nargin<5)
    linkLengths=[50,90,20,120,50.65,70,28.2];
  end
  a=linkLengths(1);
  b=linkLengths(2);
  c=linkLengths(3);
  d=linkLengths(4);
  e=linkLengths(5);
  sin_theta11=sin(theta1(1)); % these come from the IK
  sin_theta12=sin(theta1(2)); % these come from the IK
  sin_theta13=sin(theta1(3)); % these come from the IK
  cos_theta11=cos(theta1(1)); % these come from the IK
  cos_theta12=cos(theta1(2)); % these come from the IK
  cos_theta13=cos(theta1(3)); % these come from the IK
  cos_theta31=(py*cos(phi1)-px*sin(phi1))/d;
  cos_theta32=(py*cos(phi2)-px*sin(phi2))/d;
```

```
cos_theta33=(py*cos(phi3)-px*sin(phi3))/d;
sin_theta31=sqrt(1-cos_theta31^2);
sin_theta32=sqrt(1-cos_theta32^2);
sin_theta33=sqrt(1-cos_theta33^2);
sin_theta121=(pz-b*sin_theta11)/(d*sin_theta31-c);
sin_theta122=(pz-b*sin_theta12)/(d*sin_theta32-c);
sin_theta123=(pz-b*sin_theta13)/(d*sin_theta33-c);
cos_theta121=sqrt(1-sin_theta121^2);
cos_theta122=sqrt(1-sin_theta122^2);
cos_theta123=sqrt(1-sin_theta123^2);
% Jx - 3x3
Jx(1,1)=-cos_theta31*sin(phi1)+cos_theta121*sin_theta31*cos(phi1);
Jx(2,1)=-cos_theta32*sin(phi2)+cos_theta122*sin_theta32*cos(phi2);
Jx(3,1)=-cos_theta33*sin(phi3)+cos_theta123*sin_theta33*cos(phi3);
Jx(1,2)=cos_theta121*sin_theta31*sin(phi1)+cos_theta31*cos(phi1);
Jx(2,2)=cos_theta122*sin_theta32*sin(phi2)+cos_theta32*cos(phi2);
Jx(3,2)=cos_theta123*sin_theta33*sin(phi3)+cos_theta33*cos(phi3);
Jx(1,3)=sin_theta121*sin_theta31;
Jx(2,3)=sin_theta122*sin_theta32;
Jx(3,3)=sin_theta123*sin_theta33;
Jq=zeros(3,3);
Jq(1,1)=b*(sin_theta121*cos_theta11-sin_theta11*cos_theta121)*s
  in_theta31;
Jq(2,2)=b*(sin_theta122*cos_theta12-sin_theta12*cos_theta122)*
  sin_theta32;
Jq(3,3)=b*(sin_theta123*cos_theta13-sin_theta13*cos_theta123)*
```

```
    sin_theta33;

  Jaa=Jq\Jx;

  %% stiffness calculations
  Stm=100*(Jaa')*Jaa;

  %St11=trace(Stm);  % total stiffness (x + y +z)
  St11=(Stm(1,1)+Stm(2,2))/2;  % lateral stiffness (x + y)
  stiffness = St11;

  %% dexterity calculations
  lam1=max(eig(Stm));
  lam2=min(eig(Stm));
  dexterity=real(2*sqrt(lam1/lam2));
  errorFlag=0;
end
```

Appendix C

This code calculates the workspace envelope for a robot of given link lengths.

The code needs to be enclosed in a file called objF.m

```
function y = objF(x,w,stepSize,plotVolume)
  if(nargin<4)
    plotVolume=0;
  end
  if(nargin<3)
    stepSize=10;
  end
  if(nargin<2)
    w=[0.01,0.99];
  end


  count=0;
  SS = zeros(100,3);
  stiffness = zeros(100,1);
  dexterity = zeros(100,1);
  del=stepSize;
  % mechanism dimensions (general constants) in mm
  a=50;       %distance from origin to first r joint
  b=90;       %length of first link
  c=27.7;     %new length, +ve is extra parallel length
  d=450;%120; %length of parallel linkage
  e=50.65;    %length from end point to r joint on end effector
  %length of the link that connects the servo
```

```
%to the first delta link
g=70;
s=28.2;     %length of the servo horn
if(nargin>0)
  c=x(1,1);
  d=x(1,2);
end
linkLengths=[a,b,c,d,e,g,s];
for pz=0:del:500
  for py=-200:del:200
    for px=-200:del:200
      %inverse kinematics
      [theta thetaServo error1]=
          deltaInverse(px,py,pz,linkLengths);
%if the point is valid, add it to the list and calculate
%the stiffness
       if (error1==0)
         %calculate stiffness and dexterity
         [newStiffness,newDexterity,error2]=

    deltaStiffnessAndDexterity(px,py,pz,theta,linkLengths);
          if(error2==0)
            count=count+1;
            stiffness(count,1)=newStiffness;
            dexterity(count,1)=newDexterity;
            SS(count,:)= [px, py, pz];
```

```
        end
      end
    end
  end
end
%% calculate volume using alphavol
%(a function from the MATLAB user created code base
if (count>4)
  workspaceVolume=alphavol(SS,del*3,plotVolume);
else
  workspaceVolume=0;
end

%% calculate average stiffness
if (count>4)
  stiffAvg=0;
  m=0;
  for k=1:1:size(stiffness(:,1))
    stiffAvg=stiffAvg+stiffness(k,1);
    m=m+1;
  end
  stiffAvg=stiffAvg/m;
  averageStiffness=stiffAvg;
else
  averageStiffness=0;
end
%% calculate dexterity for robot
```

```
    if (count>4)
      averageDexterity=sum(dexterity);
    else
      averageDexterity=100;
    end
    %% calculate weighted output
    y=-0.0000019*workspaceVolume+0.000015*averageDexterity
        -0.24*averageStiffness;
end
%%%
```

Appendix D

Arduino Code

```
/*****************************************
 ** Thesis Delta robot v01  **
 ** This program controls 3 servo motors connected
 **  to a new Delta configuration robot.
 **
 ** The servos are connected to the delta arms using
 **  4-bar linkages, so the full 180 degrees of the
 **  servo motor translates to 90 degrees of movement
 **  of the delta arm. This increases the overall
 **  resolution and motor stiffness.
 **
 ** By: Jonathan Hodgins  ** Date: Feb 20, 2012
 *****************************************/
#include <Servo.h>
#include <math.h>
/***********************
 **inverse kinematic calculation variables
 ***********************/
// Functions
void deltaInverse(double x0, double y0, double z0,


    double &calcTheta1,double &calcTheta2, double &calcTheta3,
    int &errorFlag);

int deltaSingleArm(double x0, double y0, double z0,
```

```
    double &thetaServo);

//main variables
double x=0, y=0, z=100;      //end effector coordinates
double theta1,theta2,theta3; //servo arm angles
int error=0;                 //used to catch errors in placement
const double pi=M_PI;
const int debugOn=0;
/**********************
 ** Servo variables
 **********************/
Servo servo1;
Servo servo2;
Servo servo3;
int servoPin[3] = {
  9,10,11};           //pins the servos are attached to
double angle[3] = {
  0,0,0};             // variable to store the servo position
 double servoOffset[3] = {
  -3.0,-2.0,-5.0};    //centers the 0 position of the servo
void setup()
{
  servo1.attach(servoPin[0]);
  servo2.attach(servoPin[1]);
  servo3.attach(servoPin[2]);
  Serial.begin(9600);
}
/*********************************
```

```
 **Main loop
 *********************************/
void loop()
{
  int error=0;
  interrupts();
  double r=15;
  double stepSize=0.1;
  do
  {
    for(double i=0;i<M_PI*2*3;i=i+stepSize)
    {
      x=sin(i)*r;
      y=cos(i)*r;
      z=i*12/M_PI+70;
      deltaInverse(x,y,z,angle[0],angle[1],angle[2],error);
      updateSerial();
      if(error==0)
      {
        updateServo();
        delay(1);
      }
    }
    for(double i=M_PI*2*3;i>0;i=i-stepSize)
    {
      x=sin(i)*r;
      y=cos(i)*r;
```

```
        z=i*12/M_PI+70;
        deltaInverse(x,y,z,angle[0],angle[1],angle[2],error);
        updateSerial();
        if(error==0)
        {
          updateServo();
          delay(1);
        }
      }
    }
    while(1);//loop forever
    for(;;){}//stop the program if it escaped the above loop
}
/*******************************
 ** calculates inverse kinematics
 ** of new delta robot
 *******************************/
// returned status: 0 = OK, <0 = invalid position
void deltaInverse(double x0, double y0, double z0,

    double &calcTheta1, double &calcTheta2, double &calcTheta3,
    int &errorFlag)
{
  //trig constants used to rotate coordinate system
  double cos120=cos(120*pi/180);
  double sin120=sin(120*pi/180);
  double theta1,theta2,theta3;
```

```
int errorStatus=0;
if(debugOn)
{
  Serial.println("");
  Serial.print(x0);
  Serial.print(" ");
  Serial.print(y0);
  Serial.print(" ");
  Serial.print(z0);
  Serial.print("\t");
}
//rotate coords to +120 deg and calculate second arm
if (errorStatus == 0)
{
  errorStatus = deltaSingleArm(x,y,z,theta1);
}
//rotate coords to +120 deg and calculate second arm
if (errorStatus == 0)
{
  errorStatus =

  deltaSingleArm(x*cos120+y*sin120,y*cos120-x*sin120,z,theta2);
}
//rotate coords to -120 deg and calculate third arm
if (errorStatus == 0)
{
  errorStatus=
```

```
    deltaSingleArm(x*cos120-y*sin120,y*cos120+x*sin120,z,theta3);
  }
  //update the output only if everything works
  if (errorStatus == 0)
  {
    errorFlag=0;
    calcTheta1=theta1;
    calcTheta2=theta2;
    calcTheta3=theta3;
  }
  else
  {
    if(debugOn)
    {
      Serial.print(errorStatus);
    }
    errorFlag=errorStatus;
    calcTheta1=0;
    calcTheta2=0;
    calcTheta3=0;
  }
}
/*******************************
 ** inverse kinematics:
 **(x0, y0, z0) -> servo angle
 ** called by deltaInverse 3 times
```

```
 **
 ** if 0 is returned all is well
 ** if <0 is returned it is an invalid position
 ******************************/
int deltaSingleArm(double x0, double y0, double z0,
    double &thetaServo)
{
  double theta1,theta2;
  int errorAlert=0;
  //mechanism dimensions (general constants) in mm
  double a=50;   //distance from origin to first r joint
  double b=90;   //length of first link
  double c=20;   //new length, +ve is extra parallel length
  double d=120;  //length of parallel linkage
  double e=50.65;//length from end point to r joint on end effector
  //length of the link that connects
  //the servo to the first delta link
  double g=70;
  double s=28.2; //length of the servo horn
  double Ps[3]={63,0,-40}; //position of the servo joint
  double Pa[3]={a,0,0};    //position of the r joint on the base
  // Variables used during calculations
  double f=0,dCalc=0,aCalc=0,hCalc=0;
  double Pee[3]={x0,y0,z0};  //position of the end effector
  //position of the r joint on the end effector
  double Pd[3]={e,0,50};
  double Pb[3]={0,0,0};      //position where the two links meet
```

```
double Pg[3]={0,0,0};      //point on the end of the servo horn
//point where the servo link meets the base delta arm  double
Pm[3]={0,0,0};
//Physical joint limits
double maxDeltaArmAngle=80;
double minDeltaArmAngle=0;
double maxServoAngle=(90.0-3.0); //max angle from center
double minServoAngle=-(90.0-3.0); //min angle from center
Pa[0]=a; Pa[1]=0; Pa[2]=0;
//position of the r joint on the end effector
Pd[0]=x0+e; Pd[1]=y0; Pd[2]=z0;
//check if f is a real distance and not horizontal
if(abs(Pd[1])>(d*0.95))
{
  errorAlert=-1;
  thetaServo=0;
  return errorAlert;
}
//calculate projected length of parallel link
f = sqrt((d*d)-(Pd[1]*Pd[1]))-c;
//calculate distance between Pa and Pd
dCalc=sqrt((Pd[0]-Pa[0])*(Pd[0]-Pa[0])+
  (Pd[2]-Pa[2])*(Pd[2]-Pa[2]));
//if it is an invalid position
if(dCalc>b+f || dCalc<abs(f-b) || dCalc==0)
{
  errorAlert=-2;//return error 2
```

```
  thetaServo=0;
  return errorAlert;
}
aCalc=(b*b-f*f+dCalc*dCalc)/(2*dCalc);  //see notes
hCalc=sqrt(b*b-aCalc*aCalc);            //see notes
//calculate x for where 2 links meet
Pb[0]=Pa[0]+aCalc*(Pd[0]-Pa[0])/dCalc+hCalc*(Pd[2]-Pa[2])/dCalc;
Pb[1]=0;    //calculate y for where 2 links meet
//calculate z for where 2 links meet
Pb[2]=Pa[2]+aCalc*(Pd[2]-Pa[2])/dCalc+hCalc*(Pd[0]-Pa[0])/dCalc;
if(Pb[2]>Pd[2])
{
  errorAlert=-3;
  thetaServo=0;
  return errorAlert;
}
//angle of the delta robot arm
theta1=atan2(Pb[2]-Pa[2],Pb[0]-Pa[0])*180/M_PI;
if(debugOn)
{
  Serial.print(theta1);
  Serial.print(" ");
}
if(theta1<minDeltaArmAngle || theta1>maxDeltaArmAngle)
{
  errorAlert = -4;//return error 4
  thetaServo=0;
```

```
    return errorAlert;
  }
  //extra servo link calculations begin here (same method as above,
  //different values)
  Pm[0]=(Pb[0]-Pa[0])/2+Pa[0];
  Pm[1]=0;
  Pm[2]=(Pb[2]-Pa[2])/2+Pa[2];
dCalc=sqrt((Pm[0]-Ps[0])*(Pm[0]-Ps[0])+(Pm[2]-Ps[2])*(Pm[2]-Ps[2]));
//if it is an invalid position
 if(dCalc>g+s || dCalc<abs(g-s) || dCalc==0)
  {
    errorAlert = -5;//return error 5
    thetaServo=0;
    return errorAlert;
  }
  aCalc=(s*s-g*g+dCalc*dCalc)/(2*dCalc);//see notes
  hCalc=sqrt(s*s-aCalc*aCalc);//see notes
//calculate x for where servo horm meets link
Pg[0]=(Ps[0]+aCalc*(Pm[0]-Ps[0])/dCalc)+(hCalc*(Pm[2]-Ps[2])/dCalc);
//calculate y for where servo horm meets link
Pg[1]=0;
//calculate z for where servo horn meets link
Pg[2]=(Ps[2]+aCalc*(Pm[2]-Ps[2])/dCalc)-(hCalc*(Pm[0]-Ps[0])/dCalc);
  //make sure z is valid with check below (2 possible solutions)
  double temp1=

    sqrt((Pg[0]-Ps[0])*(Pg[0]-Ps[0])+(Pg[2]-Ps[2])*(Pg[2]-Ps[2]));
```

```
  double calcTolerance=0.1;
  //change it if the code guessed wrong
  if(temp1>s+calcTolerance || temp1<s-calcTolerance)
  {
//calculate z for where servo horn meets link
Pg[2]=(Ps[2]+aCalc*(Pm[2]-Ps[2])/dCalc)+(hCalc*(Pm[0]-Ps[0])/dCalc);
  }
  //calculate servo theta
  theta2=atan2(Pg[2]-Ps[2],Pg[0]-Ps[0])*180/M_PI;
  if(debugOn)
  {
    Serial.print(theta2);
    Serial.print(" ");
  }
  if(theta2<minServoAngle || theta2>maxServoAngle)
  {
    errorAlert = -6;//return error 6
    thetaServo=0;
    return errorAlert;
  }
  errorAlert = 0;   thetaServo=theta2;//returns the servo angle
  return errorAlert;
}
void updateServo(void)
{
  servo1.write(-angle[0]+servoOffset[0]+90);
  servo2.write(-angle[1]+servoOffset[1]+90);
```

```
  servo3.write(-angle[2]+servoOffset[2]+90);
}
void updateSerial(void)
{
  serialOut();
}
void serialOut(void)
{
  Serial.print(" x:");
  if(x>=0)
    Serial.print("+");
  Serial.print(x);
  if(abs(x)<100)
    Serial.print("0");
  if(abs(x)<10)
    Serial.print("0");
  Serial.print(" y:");
  if(y>=0)
    Serial.print("+");
  Serial.print(y);
  if(abs(y)<100)
    Serial.print("0");
  if(abs(y)<10)
    Serial.print("0");
  Serial.print(" z:");
  if(z>=0)
    Serial.print("+");
```

```
    Serial.print(z);
      if(abs(z)<100)
      Serial.print("0");
    if(abs(z)<10)
      Serial.print("0");
    Serial.print("\t t1:");
    Serial.print(angle[0]);
    Serial.print("\t t2:");
    Serial.print(angle[1]);
    Serial.print("\t t3:");
    Serial.print(angle[2]);
      Serial.println("");
}
```